

c o n f e r e n c e

proceedings

**The Tenth Systems
Administration Conference
(LISA '96) Proceedings**

*Chicago, Illinois**September 29 – October 4, 1996*

Co-sponsored by
**The USENIX Association and
SAGE, the System Administrators Guild**



The UNIX® and Advanced
Computing Systems Professional
and Technical Association



For additional copies of these proceedings contact

USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710 USA
Telephone: 510-528-8649

The price is \$30 for members and \$38 for nonmembers.

Outside the U.S.A. and Canada, please add
\$12 per copy for postage (via air printed matter).

Past USENIX Large Installation Systems Administration Workshop
and Conference Proceedings (price: member/nonmember)

Large Installation Systems Admin. I Workshop	1987	Philadelphia, PA	\$4/\$4
Large Installation Systems Admin. II Workshop	1988	Monterey, CA	\$8/\$8
Large Installation Systems Admin. III Workshop	1989	Austin, TX	\$13/\$13
Large Installation Systems Admin. IV Conference	1990	Colorado Spgs, CO	\$15/\$18
Large Installation Systems Admin. V Conference	1991	San Diego, CA	\$20/\$23
Systems Administration VI Conference	1992	Long Beach, CA	\$23/\$30
Systems Administration VII Conference	1993	Monterey, CA	\$25/\$33
Systems Administration VIII Conference	1994	San Diego, CA	\$22/\$29
Systems Administration IX Conference	1995	Monterey, CA	\$30/\$38

Outside the U.S.A. and Canada, please add \$10
per copy for postage (via air printed matter).

Copyright © 1996 by The USENIX Association. All rights reserved.

This volume is published as a collective work.

Copyright to these works is also retained by the author.

Permission is granted for the noncommercial reproduction of the
complete work for educational or research purposes.

ISBN 1-880446-81-2

AFS is a registered trademark of Transarc Corporation.

AIX is a registered trademark of IBM.

AdvanceStack is probably a trademark of Hewlett Packard.

Alpha is a registered trademark of Digital Equipment Corp.

BSD/OS is a trademark Berkeley Software Design.

Cisco is a registered trademark of Cisco Systems, Inc.

DCE and OSF/1 are registered trademarks of the Open
Software Foundation.

DEC, VMS, and DEC UNIX are trademarks of Digital
Equipment Corp.

Ethernet is a trademark of Xerox.

HP-UX is a trademark of Hewlett Packard.

Microsoft, Windows, Windows NT, Windows 95, Snapshot,
Query, Multiprotocol Filer, MS-DOS, Excel, and Data
ONTAP are trademarks of Microsoft Corp.

NEARnet is a trademark of BBN Systems and Technologies.

NETSYS Technologies is a trademark of NETSYS
Technologies, Inc.

NetApp and Network Appliance are trademarks of Network
Appliance Corp.

Netscape is a registered trademark of Netscape
Communications, Inc.

OpenDist is a trademark of Siemens AG.

Pentium is a registered trademark of Intel, Inc.

Qmail is a trademark of Internet One, Inc.

Razor is a trademark of Tower Concepts, Inc.

SGI and IRIX are registered trademarks of Silicon Graphics,
Inc.

SPARCstorage is a trademark of SPARC International, Inc.

Seagate, Hawk, and Barracuda are trademarks of Seagate.

SeeLog and SeeSoft are trademarks of AT&T Bell Labs.

Sprint is a trademark of Sprint.

Sun, UltraSparc, SunOS, SparcStation, Solstice DiskSuite,

Solaris, SPARCstation, SPARCserver, PC-NFS,

Openwin, OpenWindows, OnLine: DiskSuite, NIS,

NFS, Java, XIL are trademarks of Sun Microsystems,
Inc.

Sybase, Transact-SQL, SQL, and Open Client DB-Library
are trademarks of Sybase, Inc.

UNIX is a registered trademark in the United States and
other countries, licensed exclusively through X/Open
Company Limited.

VxVM is a registered trademark of Veritas Software.

USENIX acknowledges all trademarks appearing herein.

♻️ Printed in the United States of America on 50% recycled paper, 10-15% post consumer waste.

USENIX Association

**Proceedings of the Tenth
Systems Administration Conference
(LISA X)**

**September 29-October 4, 1996
Chicago, IL, USA**

TABLE OF CONTENTS

Acknowledgments	iv
Preface	v
Author Index	vi

Opening Remarks

Wednesday (9:00-10:30 am) Chairs: Helen E. Harrison & Amy K. Kreiling

Distributed Systems Administration

Wednesday (11:00-12:30 pm) Chair: Pat Wilson

Priv: Secure and Flexible Privileged Access Dissemination	1
<i>Brian C. Hill, University of California, Davis</i>	
The Igor System Administration Tool	9
<i>Clinton Pierce, Decision Consultants Inc.</i>	
Centralized Administration of Distributed Firewalls	19
<i>Mark Miller & Joe Morris, Bell Atlantic</i>	

Account Management

Wednesday (2:00-3:30 pm) Chair: E. Scott Menter

Shuse: Multi-Host Account Administration	25
<i>Henry Spencer, SP Systems</i>	
The Design and Implementation of a Network Account Management System	33
<i>J. Archer Harris, James Madison University; Gregory Gingerich, Bell Atlantic</i>	
UNIX Host Administration in a Heterogeneous Distributed Computing Environment	43
<i>Gregory S. Thomas, James O. Schroeder, Merrilee E. Orcutt, Desiree C. Johnson, Jeffrey T. Simmelink, & John P. Moore, Pacific Northwest National Laboratory</i>	

Visualization in Systems Administration

Wednesday (4:00-5:00 pm)

Chair: Helen E. Harrison

Visualizing Huge Tracefiles with Xscal	51
<i>Alva L. Couch, Tufts University</i>	
Using Visualization in System and Network Administration	59
<i>Doug Hughes, Auburn University</i>	

Tools

Thursday (9:00-10:30 am)

Chair: Paul Evans

How to Avoid Learning Expect -or- Automating Automating Interactive Programs	67
<i>Don Libes, National Institute of Standards and Technology</i>	
An LPD for the 90s	73
<i>Mark Fletcher, SAS Institute Inc.</i>	
RUST: Managing Problem Reports and To-Do Lists	81
<i>Craig Ruefenacht, University of Utah</i>	

Networking

Thursday (11:00-12:30 pm)

Chair: Pat Parseghian

Renumbering: Threat or Menace?	91
<i>Eliot Lear, Jennifer Katinsky, Jeff Coffin, & Diane Tharp, Silicon Graphics, Inc.</i>	
OC3MON: Flexible, Affordable, High Performance Statistics Collection	97
<i>Joel Apisdorf, k claffy (NLNR), Kevin Thompson, & Rick Wilder, MCI/vBNS</i>	
IP Multiplexing by Transparent Port-Address Translator	113
<i>Heon Y. Yeom, Jungsoo Ha, & Ilhwan Kim, Seoul National University</i>	

Sendmail

Thursday (2:00-3:30 pm)

Chair: David L. Kensiski

Many Mail Domains, One Machine: The Forwarding Mailer	123
<i>Hal Pomeranz, NetMarket/CUC International</i>	
How to Get There From Here: Scaling the Enterprise-Wide Mail Infrastructure	131
<i>Michael Grubb, Duke University</i>	
Automatic and Reliable Elimination of E-mail Loops Based on Statistical Analysis	139
<i>E. Solana, V. Baggiolini, M. Ramluckun and J. Harms, University of Geneva (Switzerland)</i>	

Toasty Cool Moose

Thursday (4:00-5:30 pm)

Chair: Bill LeFebvre

MajorCool: A Web Interface To Majordomo	145
<i>Bill Houle, NCR Corporation</i>	
The PGP Moose – Implementation and Experience	155
<i>Greg Rose, Qualcomm Australia</i>	
The Brave Little Toaster Meets Usenet	161
<i>Karl L. Swartz, Network Appliance</i>	

Software Distribution #1

Friday (9:00-10:30 am)

Chair: Pat Parseghian

A Simple Caching File System for Application Serving	171
<i>John D. Bell, Ford Motor Co.</i>	
Automating the Administration of Heterogeneous LANs	181
<i>Michael Fisk, New Mexico Institute of Mining and Technology</i>	
PC Administration Tools: Using Linux to Manage Personal Computers	187
<i>Jim Trocki, American Cyanamid Company</i>	

Software Distribution #2

Friday (11:00-12:30 pm)

Chair: Elizabeth Zwicky

Abstract Yourself With Modules	193
<i>John L. Furlani, Sun Microsystems, Inc.; Peter W. Osel, Siemens Components, Inc.</i>	
SLINK: Simple, Effective Filesystem Maintenance Abstractions for Community-Based Administration	205
<i>Alva L. Couch, Tufts University</i>	
Managing and Distributing Application Software.	213
<i>Ph. Defert, E. Fernandez, M. Goossens, O. Le Moigne, A. Peyrat, I. Reguero, CERN, European Laboratory for Particle Physics</i>	

The Future of Systems Administration

Friday (2:00-3:30 pm)

Chair: Amy K. Kreiling

A New Twist on Teaching System Administration	227
<i>Raven Tompkins, University of Indiana</i>	
Institute White Pages as a System Administration Problem	233
<i>Jon Finke, Rensselaer Polytechnic Institute</i>	
New Fangled Phone Systems Pose New Challenges for System Administrators	241
<i>Snoopy, iXOS Software GmbH</i>	

ACKNOWLEDGMENTS

PROGRAM CO-CHAIRS

Helen E. Harrison, *SAS Institute Inc.*
Amy K. Kreiling, *SAS Institute Inc.*

PROGRAM COMMITTEE

Paul Evans, *Synopsys, Inc.*
David L. Kensiski, *Cisco Systems*
Bill LeFebvre, *Group sys Consulting*
E. Scott Menter, *Enterprise Systems Mgmt.*
Pat Parseghian, *Transmeta Corp.*
Pat Wilson, *Dartmouth College*
Elizabeth Zwicky, *Silicon Graphics, Inc.*

READERS

Steve Bellovin, *AT&T Research*
Steve Campbell, *Dartmouth College*
Srinivasan Keshav, *Cornell University*
Jim Matthews, *Dartmouth College*
Dave Presotto, *Bell Laboratories*

INVITED TALKS COORDINATORS

Rik Farrow, *Internet Security Consulting*
Kim Trudel, *Massachusetts Inst. of Technology*

WORK-IN-PROGRESS COORDINATOR

Adam Moskowitz, *Interval Research Corp.*

GURU-IS-IN COORDINATOR

Steve Simmons, *Inland Sea*

TERMINAL ROOM COORDINATOR

Gretchen Phillips, *University at Buffalo*

PROCEEDINGS PRODUCTION

Rob Kolstad, *Berkeley Software Design, Inc.*
Pennfield Jensen, *USENIX Association*
Data Reproduction

USENIX MEETING PLANNER

Judith F. DesHarnais, *USENIX Association*

USENIX EXECUTIVE DIRECTOR

Ellie Young, *USENIX Association*

USENIX SUPPORT STAFF

Colleen Biddle, *USENIX Association*
Eileen Curtis, *USENIX Association*
Diane DeMartini, *USENIX Association*
Julie Keiser, *USENIX Association*
Toni Veglia, *USENIX Association*

USENIX MARKETING DIRECTOR

Zanna Knight, *USENIX Association*

USENIX EXHIBITS COORDINATOR

Cynthia Deno, *USENIX Association*

PREFACE

In ten years the LISA conference has grown from a small workshop held in a Philadelphia hotel meeting room to a multi-day, multi-track technical conference that is a “must attend” event for system administrators from around the world. For the tenth anniversary of the LISA conference, we are proud to welcome you to Chicago and are confident that we have put together a conference that will appeal to everyone.

Our first thanks should go to our program committee who did a remarkable job of reviewing the submissions and of selecting the final papers. Another round of thanks go to those who took the time to submit an abstract. Paper submission rates for this year’s LISA conference were 50% higher than in previous years. The quality of the abstracts was once again outstanding; due to the large number received, we just couldn’t accept as many as we would have liked. We encourage everyone, both those new to LISA as well as our veteran attendees, to submit a paper next year. Without your papers, there wouldn’t be a LISA conference!

Each year, the LISA conference requires a lot of behind-the-scenes effort by a lot of people. This year was no different. Our program committee members did an excellent job of pulling together an exciting and technically challenging refereed papers program in a very short amount of time. In addition, several people helped serve as ‘readers’ to review and comment on the abstracts. With ten very informative invited talks scheduled, our invited talks coordinators, Rik Farrow and Kim Trudel, deserve our thanks as well. Also working hard for LISA 96 were our Guru-Is-In coordinator, Steve Simmons, and our Works-In-Progress coordinator, Adam Moskowitz. Many thanks to all of you.

USENIX and SAGE are fortunate to be supported by such an efficient, resourceful and helpful office staff. USENIX events such as LISA wouldn’t happen without these extremely talented individuals, and we are very appreciative of the assistance they provided us while planning for LISA 96.

Our final thanks go to Rob Kolstad. Rob has been instrumental in each and every LISA Conference – from formatting the LISA proceedings, to drawing the crowds for the final afternoon events to just being himself, a LISA conference wouldn’t be the same without you, Rob!

Thank you for attending the 1996 LISA conference!

Helen E. Harrison
Amy K. Kreiling

AUTHOR INDEX

Joel Apisdorf	97	Don Libes	67
V. Baggiolini	139	Mark Miller	19
John D. Bell	171	John P. Moore	43
k claffy	97	Joe Morris	19
Jeff Coffin	91	Merrilee E. Orcutt	43
Alva L. Couch	51, 205	Peter W. Osel	193
Ph. Defert	213	A. Peyrat	213
E. Fernandez	213	Clinton Pierce	9
Jon Finke	233	Hal Pomeranz	123
Michael Fisk	181	M. Ramluckun	139
Mark Fletcher	73	I. Reguero	213
John L. Furlani	193	Greg Rose	155
Gregory Gingerich	33	Craig Ruefenacht	81
M. Goossens	213	James O. Schroeder	43
Michael Grubb	131	Jeffrey T. Simmelink	43
Jungsoo Ha	113	Snoopy	241
J. Harms	139	E. Solana	139
J. Archer Harris	33	Henry Spencer	25
Brian C. Hill	1	Karl L. Swartz	161
Bill Houle	145	Diane Tharp	91
Doug Hughes	59	Gregory S. Thomas	43
Desiree C. Johnson	43	Kevin Thompson	97
Jennifer Katinsky	91	Raven Tompkins	227
Ilhwan Kim	113	Jim Trocki	187
O. Le Moigne	213	Rick Wilder	97
Eliot Lear	91	Heon Y. Yeom	113

Priv: Secure and Flexible Privileged Access Dissemination

Brian C. Hill – University of California, Davis

ABSTRACT

Large UNIX installations have become increasingly demanding of ever growing staffs of system administrators. Since very few system management tasks can be performed without access to root, large sites quickly face a “too many cooks” problem. The problem emerges clearly out of the numerous existing, yet only partial solutions to the problem.

Examples of broad access needs are numerous: Help desk staff need to be able to examine user's files to assist over the phone or change passwords. The ability to *su(1)* to only non-root users is also be useful. System operators need to manage print queues, kill jobs and reboot systems. Neither group, however, should necessarily need access to to pids or files owned by root, making blanket access to *kill(1)* and *cat(1)*, for example, potentially problematic. Users in a research lab need to mount cdroms. In walk-in consulting areas, securing access to commands via the privileged user's own *passwd* prevents unattended terminals from being sabotaged. In our environment at UC Davis, we have all of these groups of users and none represent core system administration staff, the only true root users. Several packages and other mechanisms address this issue, but even most of the more expensive commercial packages fail to allow privileged system access that is both secure and flexible.

Access Control Issues

Root access control and auditing/logging fall under the category host-based security. Root access control determines the conditions necessary for a user to successfully invoke a privileged command. The purpose of access control is not only to prevent or limit malicious activity, but to mitigate detrimental accidental activity as well. For example, if *kill(1)* is inadequately protected, a user might accidentally mistype the pid argument to *kill(1)* and kill an important system daemon. The usefulness of auditing and logging begins after a user has successfully launched a command.

Access restriction means are numerous. Simple methods include restricting program invocation by user, groups of users, host, and user password. More elaborate means include restrictions on time of day, point of access (tty, remote host), additional challenge passwords, arguments (system objects and their owners, numbers of arguments, options). Advanced methods usually involve system call tracing or intercepting. The methods available have largely been defined by the various existing packages, but a summary of methods can also be found in the National Computer Security Center's (NCSC) “A Guide to Understanding Discretionary Access Control in Trusted Systems” [NCSC1]. Last updated in 1987, this document is admittedly dated, but useful as reference.

The user and group restrictions are the most basic access restrictions. They are supported by all packages and methods (referred to hereafter as just packages).

The host restriction is more of a configuration convenience, allowing centralized configuration to be used across several hosts.

The password restriction is an interesting concept. It requires a user to enter his/her password to execute a privileged command. The advantage is that users do not need to be given passwords to other logins, most importantly to the root login. There are fewer passwords for a user to remember (just his/her own, generally) and it is easier to revoke privileges for user without having to change a password and distribute the new password to all the users that do need it. The unfortunate consequence is that the more users there are that have this kind of access, the more passwords that will float around that can be used to gain privileged access, since passwords can potentially be sniffed as they are typed over an insecure network. Password challenges (discussed later) and single-use passwords provide an important way to mitigate this problem.

The time restriction limits access by time of day or week, etc.. The difficulty with time restrictions is the extent to which time syntax can vary.

Point of access restrictions, which limit access by a user's location, include tty and remote host restrictions. A tty restriction can be used limit access to a given command to only the console, requiring a user to be on-site to gain privileged access. Restricting tty access can also limit batch-mode operations, when that is deemed to be undesirable, by disallowing input and or output to be redirected from or to a plain file.

Unfortunately, programs like 'expect' [Libes] will foil such restrictions by allowing batch input to be supplied under the guise of a regular tty.

Restricting access based on remote host can be done one of two ways. The most portable way is with the information from `/etc/utmp`, which can be used in a straightforward fashion, but this information is often inaccurate and, on older systems, writable by the user. This may become more viable in the future as more version of UNIX make this information non-world-writable and more difficult to alter in general. The other, much more solid method is to use the socket information behind the input file descriptor, but the kernel data structures needed to derive the remote host vary widely between operating systems. Lsof [UPurdue] reportedly contains code to accomplish this.

The time and point of access restrictions be can used to limit access to employees after business hours or away from the office [FSA2].

Password challenge mechanisms take a variety of forms. Some are as simple as requiring a second user passwd contained in password database, possibly encrypted, separate from the UNIX passwords. Others simply require additional information, including social security numbers, date of birth and mothers' maiden names. Better mechanisms include those that require the password of another user (safe-deposit box style). The best mechanisms implement single-use passwords, like DES Gold [Enigma]. Some privileged access packages support these external single-user password packages.

Explicit setting of user id to other than root, changing of the umask and setting environment variables are also ways to improve security and flexibility. A program that only needs to be run as a miscellaneous system user, should not be run as root. The print job cancelling command `cancel(1)` can also be run as user 'lp' on most systems. A command that appears in different places on two different operating systems can be configured as one command in the privileged command database, by setting the PATH only to include the two locations where the command is located. Many commands in `/usr/etc` on older UNIX systems and `/usr/sbin` on newer systems fit this profile. Setting the umask to explicitly prevent creation of world writable files as the result of a `cp(1)` command is also useful.

The restriction mechanism least utilized by packages focus on the command line arguments, including number, option clusters (also known as switches), relative argument position and, most importantly, arguments that represent system objects. Restricting the number of arguments is often useful when command or an activity cannot be tailored to totally eliminate the potential for damage. This has the effect of slowing down the accidental damage a user might be able to do, by requiring repeated interaction to accomplish a

large task, like `chown'ing` or `chmod'ing` large lists of files. The options restriction can be used to eliminate the case where a user might supply a '-R' to `chown`. A simple string comparison is not sufficient as options can usually be mixed with other options, forcing all permutations of available options for a command to be specified. Many commands will simply ignore unknown options, increasing the number of permutations that would have to be specified. More specifically, restrictions can be placed on system objects presented on the command line, like login ids, files, pids, and print jobs. Systems that implement restrictions on system objects are often referred to as 'capability-based' systems [NCSC1]. This is the most powerful and flexible of all restrictions. A user can be granted access to `kill(1)` all pids not owned by root or other important system users, like daemons. Preventing a user from specifying arguments in certain places on the command line can also be controlled with a restriction that causes other restrictions only to affect certain arguments. The typical use of this would be with commands like `mv(1)` and `cp(1)`, where the targets (which always appear at the *end* of the command line) are more vulnerable than the sources. A `cp(1)` command could be configured to allow copying *from* files owned by any user, including root, but only *to* any file and directories *not* owned by root. Any ownership(s) of destinations can be specified.

A template is a way to specify mandatory parts of a command, which accomplish a couple of odds and ends. They allow commands with long argument lists that don't vary much, like adduser programs, to be shortened to just a few arguments, simplifying procedures for privileged users. A template also allows a command to be configured to allow arguments in a specific order. If you want to restrict an adduser program to not allow "root" to be added or changed, you could specify simply that no "root" strings appear in on the commands line, but, for example, if the `gecos` field needed to contain the string "root", this would not be practical. Argument position restrictions would not work because many adduser programs, like Solaris 2.5's `useradd(1)`, allow option and corresponding arguments to be specified in any order. In this situation, a template can be created that nails the options to a particular location in the command line, requiring the user to only supply the arguments to those options, thereby allowing restriction to only be placed only on certain arguments.

Executable checksums on binaries are another safeguard in a privilege dissemination mechanism that reduce the likelihood of Trojan Horses, but the information is difficult to maintain. Anytime a binary is patched by the operating system vendor or a locally maintained binary is updated to a later version, checksum information in a checksum database must be changed.

Configuration languages/syntax also vary between packages. Some use simple tables and others

use fully BNF (Backus-Naur Form) languages. Some languages use a syntax that falls 'in between', similar to a *printcap(4)* file.

Though each type of restriction has its weaknesses, taken together, they provide some very flexible, yet secure means of disseminating privileged access. Priv attempts to just that.

Auditing

Logging of privileged activity takes many forms, from recording general details about the access, to full session capturing. The simplest information includes the user's login id, tty, current working directory, command plus arguments and time of execution. In some cases, just syslog(2/3) is used, but other packages manage their own log files. More elaborate schemes encrypt logged information and send it to a central log host. Some packages also offer full session capture capability.

While many commands can be configured securely to give access to a wide variety of users without much risk, some commands inherently remain at least partially vulnerable. Non-writing commands, like *ls*, though potentially intrusive, pose relatively little threat if configured properly. Logs of these commands might be of little interest. Commands like *cat(1)* and *kill(1)*, however, present a slightly increased threat that a system manager would probably want to monitor more carefully, but still with a simple log. In some cases, for example, it might be desirable to give experienced customer service agents full *su(1)* access to group of regular user accounts (but not root or daemon, for example), allowing more flexible privileged movement, and therefore better customer service. What a system manager depends on at that point, is a full session capture. Auditing obviously becomes a tricky task, when a privileged user has enough access to be able to erase the logs. Sending audit information to a remote log host, which itself must presumably have to have more secure access than the others, largely alleviates this problem. Encrypting logs is yet an additional measure that makes alterations nearly impossible. In the event that a log were kept locally on a system, a user would be forced to remove an entire log to cover up an illicit act, raising a red flag to system managers. Another way to handle sensitive commands is to have log information appear on the system console or other designated ttys and/or have mail sent to system managers with the log information just before command execution.

More general guidelines for auditing are available in the NCSC's "A Guide to Understanding Audit in Trusted Systems" [NCSC2].

The Packages

The simplest solution could be one using UNIX groups, where setuid copies of some key unix commands, like *cat(1)*, *kill(1)*, *lprm(1)*, and *ls(1)* could be

placed in a directory created for privileged commands with execute permission set only for certain groups of users. This solution lacks log keeping and additional protections like per-use passwords. Additionally, in an environment where users would only need certain privileges on certain systems, different group files would need to be maintained on different systems, increasing management tedium.

Some vendors offer home grown solutions that lack some of the same flexibility as the third party tools do, but most importantly, they lack cross-platform compatibility. Access Control Lists (ACL's), for example, offer a very precise mechanism for granting access to specific files to specific users. Among the limitations of ACL's is that some platforms don't have them and those that do, including SunOS 5.X, DYNIX/ptx and HPUNIX, each have slightly different implementations of ACL's; this makes it difficult to devise a single, installation-wide strategy in a heterogeneous installation. Also, ACL's are limited to files, they cannot be used to protect process ids in a similar manner. Lastly, removing users and managing global permissions is cumbersome as operations have to be performed on a per-file basis.

Though the focus here is on UNIX tools, mainframe and mainframe-like operating systems have long had native privilege dissemination mechanisms more elaborate than UNIX has ever had natively. IBM's "Resource Access Control Facility (RACF)" [IBM] on MVS is such an example, allowing read/write permissions to be granted to system objects by owner. VMS has had a privilege level mechanism since its inception that allow over 15 kinds of access for privileged use.

SuSub [TLogic] is the simplest of the commercial products discussed here. SuSub provides only simple user restriction, password protection and simple logging of commands.

Sudo [UColo] is a popular public domain that allows user, host and password restrictions. It support simple logging. Sudo's password restriction allows for a five minute (default) validation period. If another command is entered within five minutes, no password is required and the 'ticket' is updated. Sudo sends mail to a designated user whenever a attempt to invoke a command with Sudo fails. Sudo's main advantage is that it is simple to configure and supports many operating systems.

PowerBroker [FSA1] is a fairly comprehensive package that offers almost every feature mentioned except robust control over command line arguments, a significant drawback. While it does offer some command line argument control, it lacks control over command line arguments that might represent system objects.

Other solutions like Exu [RammGrubb] seem to focus more on the secure network aspect of implementing secure privilege dissemination, rather than the

dissemination details, and also more on centralized system management. Exu uses Kerberos.

Ideally, flexible access restrictions should be provided at the kernel level, at least as a drop-in module, like SeOS [Memcol]. This very complete product offers, essentially, every features mentioned. These types of products are even more expensive, however, than other commercial products, like Power Broker. They can also lead to increased system overhead as all suspect system calls by all users on a system are intercepted by a kernel 'user exit' routine. SeOS also does not yet support all major UNIX operating systems.

What all of these solutions lack, however, except SeOS, is the unity of all of their features in one package and, most importantly, robust control over command line arguments that might represent system objects, like files, process ids and print jobs, in a way that is consistent. A user that had kill privileges could kill anything including sending a TERM signal to init causing a system run level change to single user. A user that could cat files, could examine anything on the system. The packages and other mechanisms mentioned address the basics and some more elaborate aspects of privileged access, but even the more expensive commercial packages fail to allow privileged system access that is both secure *and* flexible.

Priv offers a highly flexible approach to disseminating access to privileged commands that is secure and easy to manage in a large installation, on a variety of UNIX platforms. It ties together most of the features of other packages and adds control over command line arguments. The basic strategy is such:

1. A privileged user runs a command with priv:

```
$ priv ls ~smith/
```

2. Priv looks up the command in a policy description file or NIS map and retrieves a restriction list.
3. Priv checks a series of conditions and/or requirements in the restriction list that must be satisfied in order to execute a command. This might involve tests on user-supplied arguments, password prompting, resetting of uid or gid, user and host checks, and more.
4. A syslog entry is made and a system console log is made.
5. A final command is assembled and executed.

A command description file contains per-command descriptions of restrictions and conditions that must be passed before a command can be executed. This file uses a format similar to a printcap file and can be accessed as a file or via NIS. Priv supports all of the following features on a per-command basis:

- Inclusive or exclusive restrictions – Conditions containing lists of items can be inclusive or exclusive.
- Restriction by user – Commands can be restricted by user, group or netgroup of users.

- Restriction by host – Commands can be restricted by host or netgroup of hosts.
- Restriction by operating system type – Commands can be restricted by operating system of current system. This makes it easier to tailor one command across platforms.
- Password protection & double password protection – Passwords can be required to execute commands. Additionally, passwords for additional neighboring privileged users, like coworkers in neighboring cubicles, can be required to add safe-deposit box-style security to command execution. Support for several vendor shadow passwd implementations is also included. This is useful for a consulting environment where one consultant would need the acknowledgment of another to execute a sensitive command like 'passwd' or 'chown'.
- Syslog(2/3) logging options – Syslog facility and level can be tailored for each command.
- Session capture – Capture session dialogue for later review.
- Setting uid and gid to other than root – Commands are executed as user root and group 0 by default, but this can be changed to any user and/or group specified.
- Setting umask – Umask can be set to specific value.
- Setting environment variables – Environment variables can be set, like PATH. This also enables single command configuration across platforms.
- Argument range – Conditions that affect arguments can be limited to certain arguments or one or more ranges of arguments specified by position numbers.
- Configuration conjunctions – "or" and "and" can be used to join multiple restrictions under one command. See below.
- Built in kill, su, cd and pwd commands – Built in commands increase execution speed and add execution behavior consistency when vendor versions of commands, like su, behave differently from vendor. Priv will also simulate cd and pwd.
- Final command rewriting template – This allows parts of a command to already be specified, taking additional arguments supplied by the users only as indicated by macros in the command rewriting template.

The most powerful aspect of priv is its robust ability to restrict arguments to commands. This is where all other well known packages fall short. Arguments can be treated as system objects – pids, files and login names.

- File ownership – Arguments that are files owned by a certain user or users in a netgroup can be restricted by limiting access or excluding access to files they owned. For files that

don't exist (i.e., the destination of a cp command), the ownership of the directory is used.

- Files – Arguments can be taken as filenames. The inodes and devices of file arguments and files listed in this restriction are compared. If this were done as string comparison, symlinks and other tricks could be used.
- Processes ownership – same as file ownership, but for pids.
- Strings – Arguments can be subject to inclusive or exclusive string, substring and regular expression comparisons.
- Login names – Arguments are resolved to uids to handle cases where uids are specified instead of login names or in cases where more than one login name maps to the same uid.

The following are some examples of the commands from a configuration file that use many of the conditions and requirements listed:

```
kill      :user=!joe,@systems: \
          :str=!.-9,!#. -kill: \
          :argv=/bin/kill,$*: \
          :or: \
          :user=joe: \
          :pid=!root,!bin,!daemon: \
          :str=!.-9,!#. -kill: \
          :argv=/bin/kill,$1:
```

Anyone from the systems netgroup, except user 'joe', may kill any job, but NOT with signal 9 (as '9' or upper or lower case '-kill'). 'joe' may only kill jobs not owned 'root', 'bin', and 'daemon', he may only kill one job at a time and he may NOT specify '-9' (or '-kill') on the commands line.

```
cp        :user=@consultants: \
          :argr=-1: \
          :fowner=!root: \
          :argv=/bin/cp,$*:
```

Consultants may cp any file, but may not cp file to any destination owned by root. The 'argr=-1' sets the argument range for later restrictions, causing the fowner parameter to only affect the last argument in the user supplied list.

```
chown     :user=@consultants: \
          :opts=!R: \
          :str=!root,!*root.\.*: \
          :os=!SunOS 5: \
          :setenv=PATH\=/usr/ucb\ : \
          :/usr/bin: \
          :argv=chown,$1:
```

Consultants may chown any file to any user except root or any root.* combination (as allowed by most chown incarnations). The 'str'ing test performs a *strcmp(3)* test on the first item in the list and a regular expression match on the second item. The chown command can be found in the supplied PATH. The '-R' option cannot be supplied (as -Rf -fR or -R) and chown cannot be run on a SunOS 5.X system.

```
su        :passwd: \
          :and: \
          :user=@systems: \
          :argv=su,$*: \
          :or: \
          :user=@consultants: \
          :login=!root,!daemon: \
          :argc=+1: \
          :argv=*su,.-,$*:
```

Require the user's passwd to su. Allow users in the systems group to su anywhere. Consultants may only su to non-root accounts. The login restriction above will not allow any argument to be specified that maps back to the uid that root owns or the uid that daemon owns. The argc restriction requires that at least one argument be specified, also preventing the user from su'ing to root by default.

Except when capture mode is required by a condition, priv does not fork, it simply execs assembled commands. Later versions will allow logging of exits codes from exec'd commands.

Priv is written in about 5000 lines of C code and compiles into roughly a 65K binary on most systems with shared libraries. Priv is written to use the largest intersection possible of system calls and library routines from system to system. It is as POSIX.1 compliant as possible. Priv goes to reasonable lengths to improve speed, for example, by eliminating unnecessary copying and comparing of strings. Priv currently builds on at least the following platforms:

- AIX 3.2-4.3
- DYNIX/ptx 4.0
- HPUX 9.0-10.0
- IRIX 4.0.5-5.3
- Linux 1.0 (various implementations)
- OSF1 2.0-3.2D (Digital UNIX)
- SunOS 4.0-5.5 (Solaris 1.0-2.5)
- Ultrix 4.0-4.4

What Priv Needs

The restriction language implementation is fully extensible, making it easy to add new conditions and restrictions to the program very easily.

Priv should offer a few other features:

- Checksum checking on executables to avoid Trojan Horse problems
- A system call trapping mechanism using UNIX *ptrace(2)* to allow secure use of vi and other programs with shell escape commands
- Restriction on time of day execution
- Support for additional challenges, like single-use password generators, etc.
- Support for additional vendor-unique shadow passwords sources
- Support for additional restriction databases and formats
- Default templates for handling general cases for faster configuration

- A graphical user interface
- Log file encryption
- Possibly enhanced language
- Control over additional system objects like print jobs
- Allow mail notification upon command execution

Other Security Issues

The mere discussion of security often heats into discussion of any weakness that package has, whether or not those weakness should be legitimately addressed by that package. The related issues require action and those that aren't deserve at least awareness. Security issues involving `setuid` programs and potential weaknesses in configuration languages must be addressed. After all, a tool specifically designed to enhance security should be the last to detract from it.

The nature of Priv's language makes it possible for sensitive commands to be configured insecurely, but this is an almost unavoidable side effect of such a flexible language. Other than the minimal checks Priv offers to avoid totally insecure configuration, it is difficult to determine how Priv, or any package like it, could even largely prevent such problems. To start, the simplest case could be examined, where a command is configured with only a user restriction. This means that Priv could require at least one restriction, and this might be reasonable, but some might argue that this isn't enough. If priv were to require any two restrictions, say user and host, it would no longer be flexible enough to tailor the wide range privileged access that it is capable of. Suffice to say, a system manager at the helm of a Priv configuration file, or any other such tool, should be well versed in security issues as s/he bears an inalienable burden of careful configuration.

Setuid programs have long taken a beating. Most of the weaknesses in setuid programs stem from logistical oversights and weak programming practices. Setuid weaknesses are pointed out, in general, in a paper presented by Karl Ramm and Michael Grubb at LISA IX [RammGrubb] and by NCSC document on access control mentioned earlier [NCSC1].

The most common logistical errors involve the file ownership and modes, especially with configuration files and temporary files. While some trusted programs have been known not to check any file permissions at all, like `rlogind(1)` on many systems, the most common problem is the race condition. A program running as a trusted user, usually root, often checks file permissions for proper ownership and mode and *then* opens a file. Intruders write programs that cycle quickly to create a symbolic link pointing to `/rhosts` (or some other vulnerable file) between the time the file permissions are being checked and the time the opening and writing begins. This is more common with temporary files that such tools would need to open in `/tmp`, which is a world writable

directory. The race condition can be mitigated by opening a file first and, and then checking permissions and ownership on the open descriptor, but many tools still do not seem to do this. The more obvious logistical issues can be addressed largely by increased conscientiousness in program design. More information about `setuid` programs and methods to test them can be found in "Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring" [UCDCS1]. The other programming weaknesses, however, are related to oversights that are less intuitive.

Inexplicit management of signals, signal handlers, inherited environment variables and maximum string lengths in programs has also created many back doors in `setuid` programs [Dilger1]. Despite the increasing visibility of these problems, `setuid` programs are still used. PowerBroker, Priv, Sudo, and SuSub all depend on the `setuid` bit. The increased awareness of secure `setuid` programming can probably be said to have made `setuid` programming more viable than it once was thought to have become.

Priv avoids race conditions and explicitly ignores all available signals that it does not use, resetting them just before executing the user-specified command. Environment variables are not affected directly, but Priv allows environment variables to be controlled via the command configuration restriction 'env'. All cases where buffer overruns might be possible have been carefully avoided. Lastly, and possibly most importantly, Priv assumes the user's id for as much of its processing as possible.

In addition to the aforementioned areas of concern, it is important to limit the focus of the discussion by reminding the reader what the discussion is not about, especially regarding network security issues.

Obviously, priv is not a `passwd` changing program like `passwd+` [Bishop], a secure `passwd` program like DES Gold [Enigma], an enhanced `login(1)` replacement like FSA's PowerLogin [FSA4], a system security integrity auditor like COPS or Tiger or a firewall. The vendors of PowerBroker, SeOS and SuSub are specialists in security packages and do offer a wide variety of such packages. Some of these are bundled with their access restriction packages discussed here, but *not* out of necessity to accompany a privileged access tool.

Most importantly, priv does not concern itself with network security. There is no encryption of traffic or attempt to authenticate any network connections. Priv only *uses* network services, like NIS, and does pretend to implement them. Concerns over network security are better left to packages like Kerberos. Priv's goal in the matter would be merely to support many database sources/engines, including more secure engines like NIS [Sun] and eventually NIS+ [Sun].

Ease Of Management

The easiest packages to manage generally allow centralized configuration for many hosts and include GUI management tool. Packages that do not allow host names to be included in the configuration, including the unix group method, SuSub and ACL's. PowerBroker, Priv and SeOS all allow configuration across hosts. SuSub, PowerBroker and SeOS provide a GUI management tool.

Performance

Performance tends not be an issue with most of these packages. The time to verify authorization and record audit information is generally quite short, usually under one tenth of one second. The only notable exception here is SeOS. Since SeOS will intercept certain system calls made by all users, whether or not a user has any authorizations for privileged action and whether or not a user attempts a privileged action, system performance will degrade slightly. Rough data provided Memco [Memco2] suggest anywhere from a 1% to a 15% drop in system performance, with the average drop being about 4%. This might be an issue for some system managers.

Price

Costs of these solutions vary widely with the features they support. The no-cost alternatives include, in order of robustness, the UNIX groups method, vendor bundled mechanisms, the partial solution of ACL's, Sudo, and Priv. The cost of commercial packages can only be roughly summarized due to the discount incentives for quantity, educational use, platform power, etc.. In order to provide a comparison, basic commercial pricing schedules for one system and thirty systems are listed. Again, discount incentives may radically change prices. Pricing listed here is subject to change in general and is intended only for comparison. In cases where platform size affected price, the least expensive platform price was used.

Prices quoted as of August 10, 1996

	1 copies	30 copies	
SuSub	\$295	\$6870	[TLogic]
PowerBroker	\$500	\$9000	[FSA3]
SeOS	\$8,000	\$80,000	[Memco3]

Conclusion

Most of these products do not go far enough and those that do are potentially pricey, depending upon the number of copies needed. The commercial product SuSub isn't even as flexible as Sudo, a limited, but seasoned public domain tool. PowerBroker is fairly comprehensive package, but lacks control over system objects. SeOS is a slick and reasonably complete product, but it is literally two orders of magnitude more expensive than the other commercial products.

Priv addresses nearly every aspect of privileged access dissemination in way that allows a high degree flexibility and security across many platforms in a large installation. Commands to accomplish very specific tasks at specific levels of access can be tailored quickly and safely. Priv combines most of features of free and commercial products into one mechanism, which is itself free. Priv is also written in such a way that it can be easily extended. The Information Technology department at UC Davis has been using priv in several units for nearly four years, as well as at a couple of private companies, a true indication of the soundness of Priv.

Acknowledgments

The late Sam McCall, Systems Programmer, U.C. Davis. Sam mostly inspired this package. Sam was always one of my best UNIX resources. This version of Priv is dedicated to him.

Dan Dorrough, Systems Programmer, U.C. Davis. Dan is still the sharpest, most solid UNIX and VMS programmer I know (if not the most old fashioned!). I doubt that I could have learned anywhere else what I learned from him about programming.

Michael Dilger, Programmer, Sun Microsystems. Mike gave me numerous pointers to help me tighten the security of the code in Priv considerably as documented above.

Dr. Chris Wee and Todd Heberlein of the Computer Security Research Laboratory and Greg Benson of Software Verification Laboratory, both of the U.C. Davis Computer Science Department. They answered all kinds of my questions about writing technical papers.

Kathryn Hemness, System Administrator, U.C. Davis. She keeps converting my shabby text man page to a *real* man page format.

Bryan Stansell, Global Networking and Computing, and Kanbiz Aghaiepour, TCSI, who worked in the trenches with me at U. C. Davis when we were students.

Dave Zavatson and Dana Drennan, System Administrators, U.C. Davis. Miscellaneous contributions.

Everyone else in Net Land that gave pointers, tips, code and other advice.

Software Availability

Retrieve the software from <ftp://ftp.ucdavis.edu/pub/unix/priv.tar.gz>, though the directory location may change in the future.

Author Information

Brian C. Hill finally got his B.S. from the University of California at Davis in 1993. He has worked part time for U.C. Davis in various systems programming and consulting roles for seven years, currently

with the Graduate School of Management and the U.C.D. GIS Center, and as an independent contractor for over five years. In addition to programming Priv, his specialties include aggressive distributed systems administration, sendmail and DNS configuration. He can be reached at bchill@ucdavis.edu.

[IBM] Resource Access Control Facility (RACF),
IBM <http://www1.ibm.link.ibm.com/HTML/SPEC/g2214102.html>

[Sun] NIS+, Sun Microsystems Incorporated.

References

- [Bishop] P] Passwd+, Dr. Matt Bishop, University of California, Davis.
- [NCSC1] Carole S. Jordan, "A Guide to Understanding Discretionary Access Control", in Trusted Systems (NCSC-TG-003), September 30, 1987, National Computer Security Center, pp. 7-11.
- [NCSC2] A Guide to Understanding Audit in Trusted Systems (NCSC-TG-001) September 30, 1987, National Computer Security Center, pp. various.
- [Dilger1] Michael Dilger, personal communication, University of California, Davis, Computer Science Department, dilger@cs.ucdavis.edu,
- [FSA1] Logging and Auditing root Actions with PowerBroker, FSA Corporation <http://www.fsa.ca/pbtech.htm>
- [FSA2] Power Broker Audit White Paper, FSA Corporation <http://www.fsa.ca/pbaudit.htm>
- [FSA3] Paul Scripko, personal communication, FSA Corporation pscripko@fsa.ca
- [FSA4] PowerLogin, FSA Corporation <http://www.fsa.ca/powerlogin.htm>
- [Memco1] SeOS, Memco Software <http://www.memco.com/products/Seos.htm>
- [Memco2] Laura Gillespie, personal communication, Memco Software, (800) 862-2602 x401, laura@memco.com.
- [Memco3] Cecilia McClaine, personal communication, Memco Software, (408) 559-8520 x125.
- [TLogic] SuSub, Technologic, Inc. <http://www.tlogic.com/susub.htm>
- [Enigma] DES Gold, Enigma Logic <http://www.safeword.com/>
- [Libes] Expect, Don Libes
- [UColo] Sudo, University of Colorado <http://www.cs.colorado.edu/~millert/sudo>
- [RammGrubb] Karl Ramm, Michael Grubb, "Exu - A System for Secure Delegation of Authority on an Insecure Network", Proceedings of the Ninth Large Installation Systems Administration Conference (LISA IX), September, 1995. <http://web.mit.edu/afs/athena.mit.edu/user/k/c/kcr/www/exu.html>
- [UPurdue] Lsof, Vic Abell, Purdue University <ftp://vic.cc.purdue.edu/pub/tools/unix/lsof>
- [UCDCS1] C. Ko, G. Fink, K. Levitt, "Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring". Proc. 10th Annual Computer Security Applications Conference Orlando, FL, 5-9 Dec. 1994, pp. 134-144.

The Igor System Administration Tool

Clinton Pierce – Decision Consultants Inc.

ABSTRACT

This paper describes the system administration tool we call Igor. Igor is a tool for administrating a large number of UNIX systems in a diverse, networked environment. Igor consists of two parts, an interactive GUI which is controlled by an operator, and a daemon which is run on the UNIX target which actually executes the commands. Igor provides very fast operation, and quick post operation analysis of the results. In normal operations we have run commands on over 600 hosts simultaneously in 60 seconds.

History

Igor was created, because in Ford's highly distributed and diverse environment, the system administrators often found themselves needing to run something on several hosts very quickly. Some examples include holiday shutdown, system re-configuration, installing patches, fixing bugs, surveying systems for patches and software usage, and emergency damage control (see Script Examples below). Normally, this would be accomplished with something like this shell script:

```
#!/bin/sh
VICTIMS="hosta hostb hostc hostd"
for cur_host in $VICTIMS
do
  rsh $cur_host "/usr/bin/cmd -arg"
done
```

This has many potential problems, not the least of which are:

- 'rsh' not being able to portably return the status of remotely executed commands. This means that in order to find out if a command worked, sometimes elaborate shell-scripting is involved.
- stdin/stderr being mixed. This makes checking for errors even more difficult on complex scripts.
- Anything beyond simple commands may involve a full-blown shell script meaning complicated rsh commands, or rcp commands, or use of an NFS filesystem to transport the scripts.
- Slow hosts can bog down operations
- Hosts whose inetd/rshd has gone to lunch cause rsh not to work properly.
- This operation is serial. If rsh hangs, your whole list of hosts goes unprocessed.
- The "batch" style of operation is quite non-satisfying. Especially after a long run, you discover a subtle bug in the script on either end, causing you to have to repeat the lengthy run.
- rsh uses whatever "mystery shell" is on the other side – and whatever environment comes with it.

A very cleverly written shell script can get around almost all of these limitations. Certainly a C program could. However, Igor gets around all of these problems and provides a neat, clean user interface for getting these kinds of jobs done. In addition, other schemes such as running administrative scripts through cron do not allow for a really interactive approach to solving these problems.

Detailed Description

Igor solves these problems by creating a fast, robust, portable, and flexible method to distribute these kinds of jobs. Igor can be described simply as a multiplexed rsh. With Igor there are two parts, a target and an controller. Security is handled through traditional "rexec"-type security (/etc/hosts.equiv and rhosts files).

Igor accepts commands from a GUI on the controlling system, and passes those commands to a set of Perl scripts which distribute the commands to the remote (target) systems. These Perl scripts maintain several remote connections simultaneously, and handle situations such as timeouts, network connection problems and terminating connections. The Perl scripts then collect the information and return it to the GUI.

The GUI itself has additional features to help the operator debug remote systems such as:

- Double-click (left) on a hostname will open an xterm on that host.
- Single-click (right) on a hostname will bring up the most recent results from that host (or group of hosts) in an editor for viewing.
- A scrollable list of hosts is always available, showing the status of the last commands run, any output or errors that resulted from the last run, and the current known state of that host (unreachable, running a job, completed, etc.)
- Information on the host-type of the various systems.

The Code

Igor is written in TCL/Tk [1] (as a wish script) and Perl 4 [2], and the target end is entirely written in Perl 4. The only prerequisites for administering a target with Igor is that the target have available to it Perl (version 4 or 5), and a host which it implicitly trusts (preferably a centrally located, tightly controlled host). The daemon which runs on the target system uses no Perl library code, and only requires that the interpreter be present. This means that all of the networking code is rolled into the daemon itself. This design was based on the decision that we wanted the daemon to rely on as little as possible to run. For example, if the perl library modules were not mounted, then we did not want the daemon disabled.

Igor works as an interactive tool, which makes it different from tools such as DSmit for AIX [5] and easier to setup than tools such as Systel [3]. Igor allows a skilled operator to write shell scripts or perl scripts and have them executed very quickly.

The Controlling system's software consists of a GUI which is entirely written in *wish(1)* and a series of backend perl scripts. The GUI simply manages the list of hosts, current set of commands, current set of

regular expression matches and some tunable preferences. The rest of the controlling system's software is a set of perl scripts that produce reports on the output data (and use the regular expression data to determine if the run was successful), setup socket connections to the remote hosts; rcp, rsh and ping the remote hosts to setup the Igor daemon,

The Target System

The target is any UNIX host which runs Perl and which trusts a central host (with */rhosts* or */etc/hosts.equiv*). Igor runs as a daemon monitoring a pre-determined TCP/IP port. This daemon can either be started by the controlling system using a "spawn" function available to the operator or it can be started by conventional means, such as rc scripts.

The "spawn" function uses ping to contact the target host, rcp to move the Igord daemon script to the target system and then rsh to run the script. The script daemonizes itself and continues running in the background. The Igor daemon (Igord) then listens to the port and when a connection is made, forks and the child process receives commands from the controlling system. These commands can be shell scripts, perl scripts, or special built-in functions to send files to the

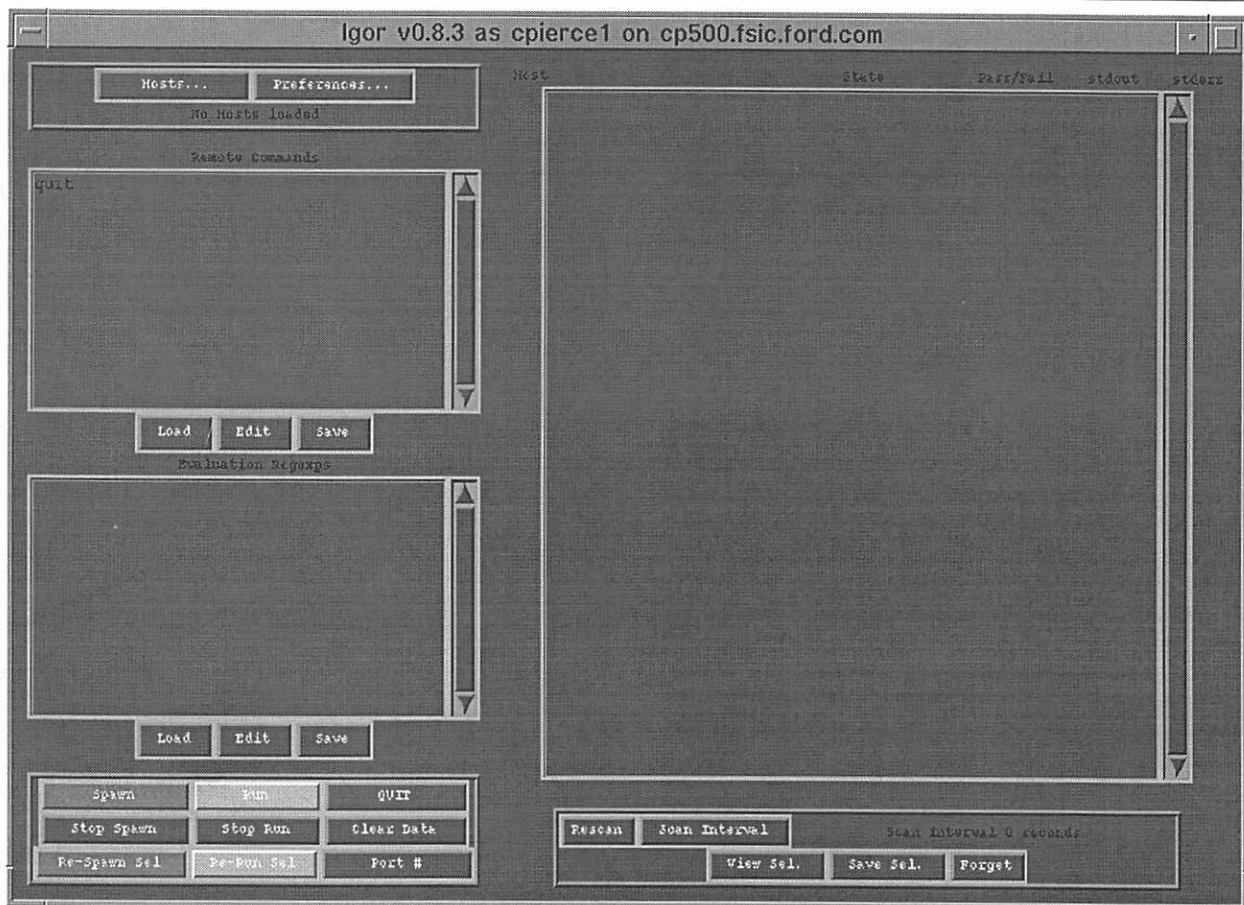


Figure 1: The graphic user interface

target. The Standard Output and Standard Error of all of the executed commands are carefully collected, and put into a boilerplate and sent back to the controlling system for analysis. The child then dies.

The preferred method of starting the daemon is to have the target system start it as part of its initialization. This way, the daemon is always available to run commands, and does not have to be “re-spawned”. Normally, at Ford, if we find a system that is not starting the daemon at boot time, we spawn a daemon on the host, and then run an Igor job to install itself on the target host, and start itself as part of the next boot.

The Controller

The operator runs Igor from a well-trusted host. This host should (if possible) have a very thick connection to the targets and should have as much CPU as you can spare; the more CPU and network, the more jobs you can run in parallel. Also, the controlling host should be able to open many sockets at once. Under certain OS's (Solaris) this requires a kernel tunable parameter to be set. The amount of resources used by the Controlling system is controlled with a “throttle” adjustable in a preferences dialog. The throttle

controls how many remote hosts will be communicated with at any one time. Setting this number high uses more resources. On a Sparc-center 1000 with 1 CPU, a throttle limit of 40 will keep the load-average of the system near 10.

On the well-trusted (Controlling) host, the operator first loads in a list of hosts to operate on. This list is simply a flat-ASCII text file, one host per line, and are loaded with a point-and-click file browser [Figure 2].

Once the hosts are loaded, you can perform “run” or “spawn” operations on those hosts or a selected subset of those hosts. Spawn is used to start Igor on the remote hosts. The host is pinged, the script is rcp'd to the host and rsh is used to get it running. Traditional BSD-style network commands are used so that the target host is almost assured to have the necessary utilities already in place to start the daemon. Once the daemon is started, you generally do not have to restart it. If a daemon is already running, and the system is “respawned”, then the new daemon will kill off the old one, and run in its place.

Once the “RUN” button is pushed the script that the operator has entered is transmitted to the remote systems, their output collected and sent back to the

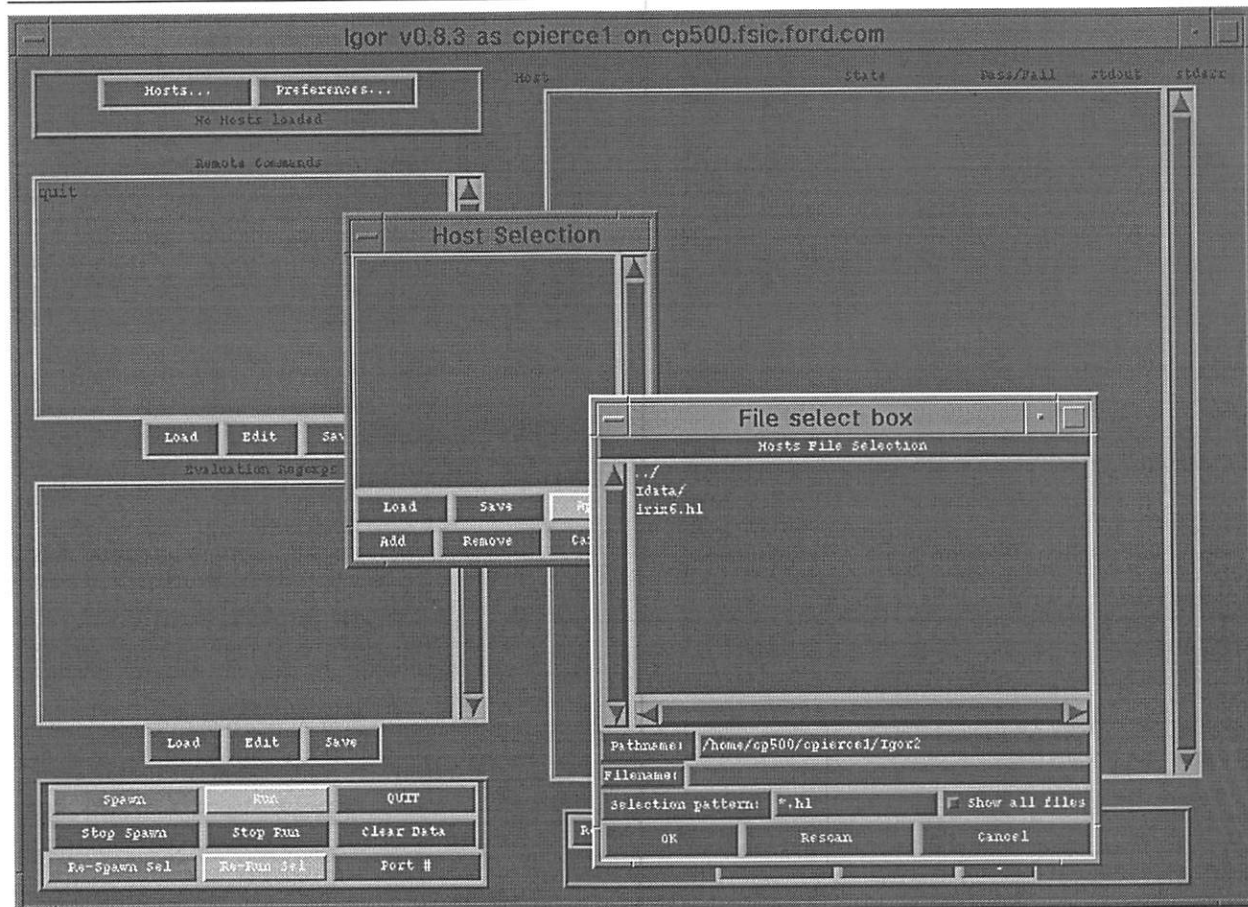


Figure 2: Point and click GUI

controlling system. For both Run and Spawn the GUI starts a back end perl script. That process forks as many times as needed to reach the “throttle” limit. Then each child takes a system name from the common hostname pool, and tries to contact the Igor on that host and execute the job. When the job is completed a particular host, the child grabs another hostname from the pool and starts again. Each of the communication processes will timeout if necessary, using a value set in the preferences dialog of the GUI, and then take another host from the pool.

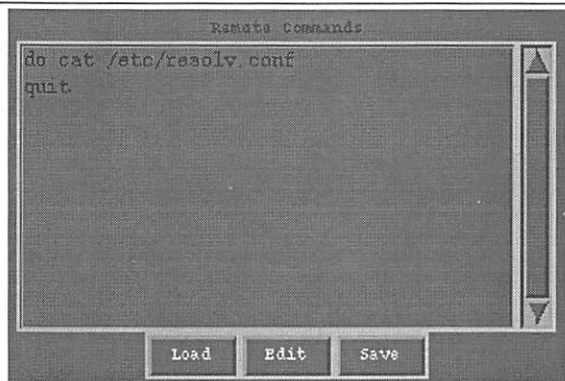


Figure 3: Script

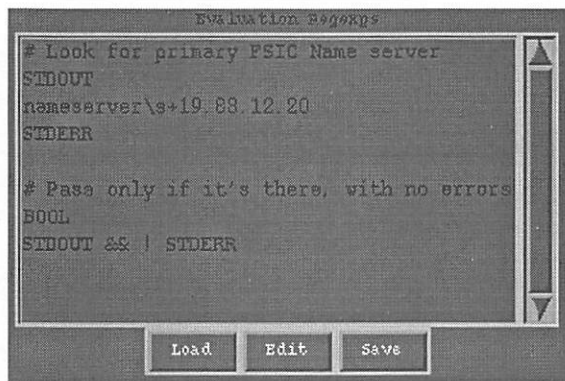


Figure 4: Regular expression list

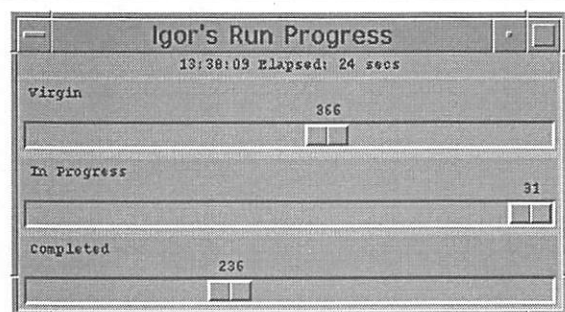


Figure 5: Progress indicator

The backend scripts and the GUI operate independently of each other. The GUI simply starts the backend scripts and then can retrieve their output by looking in a hard-wired subdirectory (./ldata) for results from each host. If the user requests another job

be run while the first job is still running, that isn't a problem. Another set of backend scripts are started, and the results are left in the same subdirectory. If for some reason the GUI needs to communicate with the backend scripts (to abort a job, for example) either token files are left in a common area which the scripts look for occasionally, or a specialized perl script can communicate back and forth between the GUI and the backend scripts using other IPC mechanisms.

Controlling GUI Tour

The buttons on the GUI [Figure 1] do the following:

- “Hosts...”: Opens the Host Selection dialog [Figure 10] and allows you to add/change/load the hosts to be worked on.
- “Preferences”: Opens the User Preferences screen [Figure 11]. The fields are:
 - Throttle – Maximum number of hosts to work on at once.
 - Timeouts – How long to wait for any one host to respond to Igor's query. Once the host is contacted, the timeout is no longer in effect.
 - Editor Options – When “view selected” is picked for multiple hosts, this selects whether you want to see one host at a time (i.e., “vi hosta hostb hostc etc..”) or all of the host data concatenated.
 - Voyeur Options – The Progress Indicator can be brought up automatically when the “Run” or “Spawn” buttons are pushed. Normally the Indicators are not shown.

Preferences are stored between sessions in .igorrc

- “Spawn”: Starts Igor on the entire set of hosts.
- “Run”: Runs the current set of commands on the entire set of hosts.
- “Re-Spawn Sel”: Starts Igor on the selected hosts
- “Re-Run Sel”: Runs the current set of commands on the selected hosts.
- “Stop Spawn”: Stops a spawn in progress. Any “spawn” that is currently being tried on a host is finished first.
- “Stop Run”: Stops a run in progress. If a host is already active, the run is finished on that host.
- “Clear Data”: Clears the ./ldata directory and removes all of Igor's information on its contacted hosts.
- “Port #”: Multiple Igor's can be run on a system simultaneously. This allows you to control which one you're talking to.
- “Rescan”: Retrieves the current set of data for each host, refresh the host status window, re-apply the regular expressions to the output.

- “Scan Interval”: A rescan can be done at a regular interval. An interval of 0 stops the auto-rescan.
- “View Sel.”: Allows you to view the current data [Figure 9] retrieved for each host selected.
- “Save Sel.”: Will save the list of selected hosts to a file. This allows you to create lists of hosts split up based on the results obtained. For example, saving all of the hosts which fail a certain test so that they can be corrected later.
- “Forget”: Remove the selected hosts from the current host list.

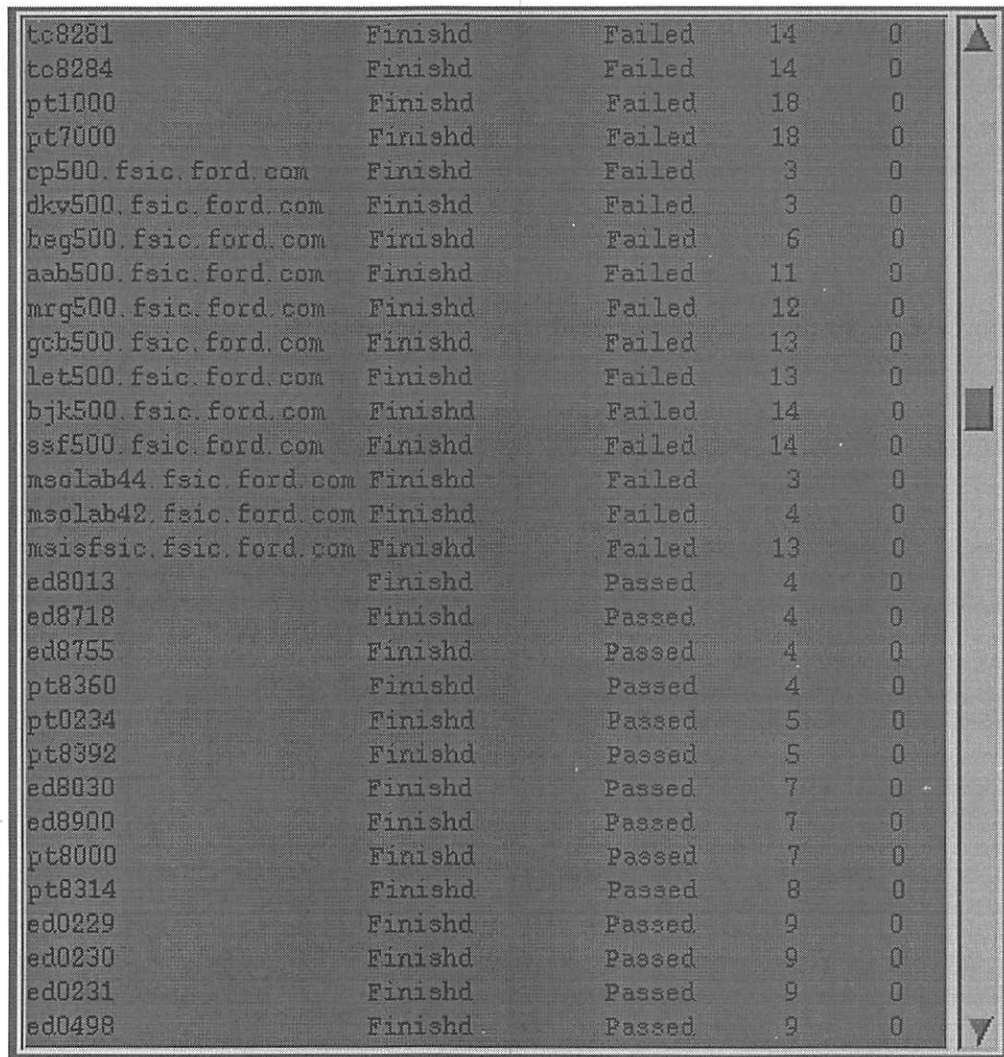
Pressing the right mouse button in Igor will bring up another panel with additional buttons [Figure 12]:

- “Archtype Sel”: Shows the architecture type of the selected hosts.
- “Watch Run”: Pops up a Progress Indicator for each Run currently in progress.
- “Watch Spawn”: Pops up a Progress Indicator for each Spawn currently in progress.

- “Kill Spawn w/Prejudice”: Stops a Spawn immediately. Does not finish the hosts currently being worked on. This can leave the remote hosts half-done. (Daemon is there, but not running for example).
- “Kill Run w/Prejudice”: Stops a Run immediately. Does not finish the hosts currently being worked on. This can leave the hosts having executed only some (or none) of the commands sent to it. Still, if you make a mistake, this button is your friend.

Igor's Security

Igor's security is based on the BSD rexec(3N)-style security of `~/rhosts` file for each operator they wish to trust, or a `/etc/hosts.equiv` file listing all of the hosts and users that they trust for Igor activity. The daemon, upon connect from a Controlling system will verify that the remote system is trusted. Having verified that, will accept



tc8281	Finishd	Failed	14	0
tc8284	Finishd	Failed	14	0
pt1000	Finishd	Failed	18	0
pt7000	Finishd	Failed	18	0
cp500.fsic.ford.com	Finishd	Failed	3	0
dkv500.fsic.ford.com	Finishd	Failed	3	0
beg500.fsic.ford.com	Finishd	Failed	6	0
aab500.fsic.ford.com	Finishd	Failed	11	0
mrg500.fsic.ford.com	Finishd	Failed	12	0
gcb500.fsic.ford.com	Finishd	Failed	13	0
let500.fsic.ford.com	Finishd	Failed	13	0
bjk500.fsic.ford.com	Finishd	Failed	14	0
ssf500.fsic.ford.com	Finishd	Failed	14	0
msolab44.fsic.ford.com	Finishd	Failed	3	0
msolab42.fsic.ford.com	Finishd	Failed	4	0
msiefsic.fsic.ford.com	Finishd	Failed	13	0
ed8013	Finishd	Passed	4	0
ed8718	Finishd	Passed	4	0
ed8755	Finishd	Passed	4	0
pt8360	Finishd	Passed	4	0
pt0234	Finishd	Passed	5	0
pt8392	Finishd	Passed	5	0
ed8030	Finishd	Passed	7	0
ed8900	Finishd	Passed	7	0
pt8000	Finishd	Passed	7	0
pt8314	Finishd	Passed	8	0
ed0229	Finishd	Passed	9	0
ed0230	Finishd	Passed	9	0
ed0231	Finishd	Passed	9	0
ed0498	Finishd	Passed	9	0

Figure 6: Results of running against various hosts

commands from the Controlling system. If the trust check fails, no commands are accepted and an error message is printed on the socket. Please note that the distributed version of Igor does not use a "trusted" port, and is for experimentation only. Simply changing the port usage on the daemon and the GUI will make Igor use a trusted port and a little more secure.

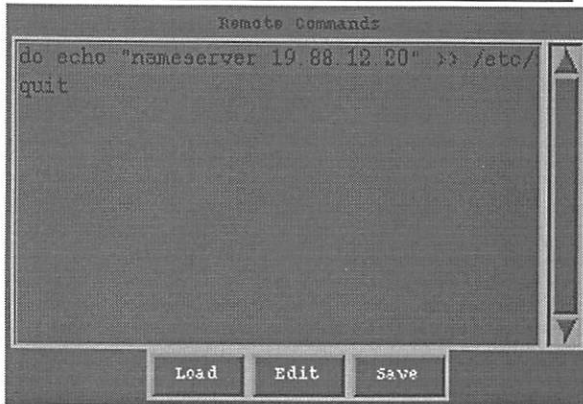


Figure 7: New set of commands

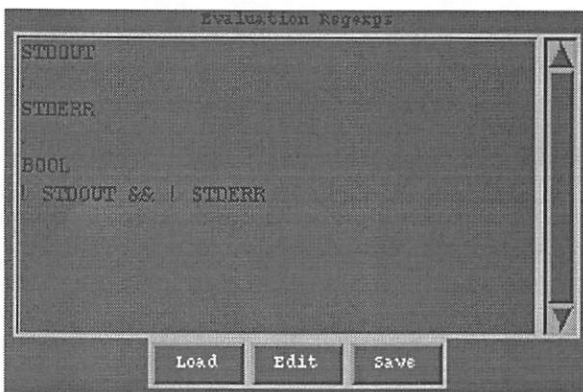


Figure 8: New set of regular expressions

This security, although old and not state-of-the-art is no less secure than what is used for rsh. Internally to Ford, this is generally adequate. Each workstation trusts a centrally located server, and our network security is handled by third party sources. Because we trust our network services and the centrally administered host, rexec security is adequate for our purposes..

There are certainly other ways to make Igor more secure. For example, using a PGP encrypted copy of the script to transmit to the remote daemons. The operator at the GUI could be queried for the encryption key. The decryption key could be located by querying another (or the same) host and having obtained the key, you could decrypt the Igor commands, ensuring that they came from the correct host.

Igor's code is fairly straightforward, and could be easily changed to accept these modifications.

Igor Scripts

The scripts that Igor runs are nothing more than a way of wrapping up shell scripts, tar files, and simple shell commands so that Igor can make sense of them at the remote side and run them. The various commands are:

- **do args** – Run args as a shell command (/bin/sh). Eventually, everything except the "do" is passed to a perl script (on the remote end) and run as a "system" command. Normal shell argument parsing will take place on the remote end. This is the most commonly used Igor command.
- **EVAL args** – Run "args" as Perl commands. This can be used to run perl commands directly by the remote daemon. Another use is to add functionality, on the fly, to the daemon by having it "eval" new functions. It can be used to add timeout capability to various Igor commands ("do"). See examples below. This is generally safe to use, because the daemon that's being modified with EVAL is simply a child of the daemon listening to the port on the remote system. Any potential defects in the child daemon do not affect its parent.
- **openfile file mode** – Open file as a "here" file with mode specified. This is one method of transmitting lengthy shell scripts to the remote system. Binary data being sent must be uuencoded because the Igor "script" exists for a while in a TCL list, which can't contain binary data. Other ways of transmitting scripts, patches, programs, etc... include using an NFS mount publicly exported Read Only from a common system, retrieving the data through an ftp script, or accessing it from a webserver with a URL. (There is a short Igor script using EVAL which can enable Igor to do HTML retrievals.)
- **closefile** – End of openfile block.
- **id** – Igor will report its version number, local hostname, remote hostname (controller), date, time, local system architecture type, and other useful information.
- **quit** – Terminate the remote Igord, transmit results. This command is REQUIRED at the end of a script, and will be inserted if you do not use one.

The commands are given to the GUI by pressing the "Edit" button in the Remote Commands window and using your favorite editor to enter commands. Pre-assembled lists of commands can be loaded using the "Load" button. Scripts are saved with a common file extension (.cmds) to distinguish them from other files.

Script Examples

To transmit a small shell script and run it:

```
openfile /tmp/fixbugs.sh 0755
#!/bin/sh
echo "Then a miracle occurs here"
install_miracle_patch
closefile
do /tmp/fixbugs.sh
do rm /tmp/fixbugs.sh
quit
```

To check disk space in / and /tmp:

```
do df /tmp /
quit
```

Using the EVAL function, some additional functionality can be added to scripts:

```
EVAL sub timeout { next MAINLOOP; }
SIG('ALRM')='timeout'; alarm(10);
do function_that_may_hang
EVAL alarm(0);
quit
```

This adds a timeout to the "function_that_may_hang". If the program doesn't return, Igor catches an alarm signal and continues executing the script after the questionable function. This required some knowledge of the innards of Igor, but these tricks are well documented.

Output Analysis

One of Igor's most important features is analyzing the output as it comes back from the remote system. In the GUI, each system is shown, with a count of the number of lines of STDOUT and

STDERR reported. Sometimes this is enough to tell if everything worked OK. Also in that window is a field labeled "Pass/Fail". This field can also be used to tag each system with a Passed/Failed status. That is done by using the Regular Expression matcher.

This area takes input in the form:

```
STDOUT
Exp1
Exp2
Expn
STDERR
Exp1
Exp2
Expn
BOOL
Boolean Expression
```

The Expressions are Perlsh regular expressions (without the //s). Slashes and special characters must be quoted. These expressions are matched against successive lines of Standard Output or Standard Error and so long as the expressions match, the associated tokens (STDOUT, STDERR) will evaluate to true. If a regex does not match, the token gets set to false. The regular expressions are associated with the token they follow. The BOOL token indicates that the next line will contain an expression that will evaluate to true or false. The Boolean Expression is a perlsh thing that is going to get EVAL'd. "STDERR" gets substituted with 1 for a match and 0 for a nonmatch "STDOUT" gets substituted with 1 for a match and 0 for a nonmatch. Depending on the outcome, the system will be marked as "passed" or "failed" in the status screen.

```
-----tcl194-----
=====REMOTE OUTPUT=====
---Opening connection
---Getting ID info
ID 0 0 V:2.0 00 17113 Tue Aug 13 13:45:03 EDT 1996 tcl194 solaris ed8030.ptoford.com
---Version=2.0
---Transmitting script
---Waiting for reply
START STANDARD ERROR
END STANDARD ERROR
START STANDARD OUTPUT
; $Id: resolv.conf,v 1.1 1994/02/28 20:34:58 jbell14 Exp $
;
; $Log: resolv.conf,v $
; Revision 1.1 1994/02/28 20:34:58 jbell14
; Initial revision
;
; Real resolver file for Domain Name Service (DNS)
; domain is ptoford.com
domain ptoford.com
search ptoford.com ptoford.com
nameserver 19.50.8.20
nameserver 19.3.230.20
nameserver 19.52.6.100
END STANDARD OUTPUT
---Reply finished
~
```

Figure 9: Sample of retrieved data

This sounds complicated, but in practice is a rather simple way of checking output. For example, to consider all systems that report something on STDOUT and nothing on STDERR as “Passed”. you could use this arrangement:

```

STDOUT
.
STDERR
.
BOOL
STDOUT && !STDERR

```

STDOUT gets set to 1 (true) if any single character is matched. STDERR gets set to 1 (true) if there's any STDERR output. If the BOOL expression evaluates to true, the system is tagged as “Passed”, otherwise “Failed”. Something more complicated could be used like this:

```

STDOUT
9[1-9]%
STDERR
.
BOOL
!(STDOUT || STDERR)

```

For the set of commands:

```
do df
```

Would report “Failed” if the “df” command reported any filesystem more than 90% full, or “df” reported anything on STDERR.

There's two other special tokens that can be used in the Boolean expression in addition to STDERR and STDOUT. These are STDERRCNT and STDOUTCNT. These represent the number of lines of output on each file descriptor. For example:

```

STDERR
.
BOOL
( STDOUTCNT > 3 ) && ( ! STDERR )

```

This would return true (passed) if there were more than three lines on stout, and nothing on stderr. This would be useful if the expected output could have a variable number of lines, but no errors should be expected.

The Pass/Fail indicators shown in the hostlists are generated every time the host list is displayed. So if you decide that a different pass/fail criteria is necessary for your hosts, you can change the regular expressions and rescan the host list. You do not need to re-run the commands on the remote hosts.

Sample Walkthrough

To demonstrate Igor's true usefulness, what follows is a walkthrough of a sample Igor session. The hypothetical problem will be adding a resolver to each workstation's /etc/resolv.conf. The operator would first login to a trusted host, and start the GUI

[Figure 1]. Next the operator can load the list of hosts to be operated on, or can enter them in [Figure 2]. First, in order to make our scripting a little easier, to find out which hosts already have the correct resolver we'll use the script shown in Figure 3. The regular expressions list is shown in Figure 4. These are loaded from a file browser.

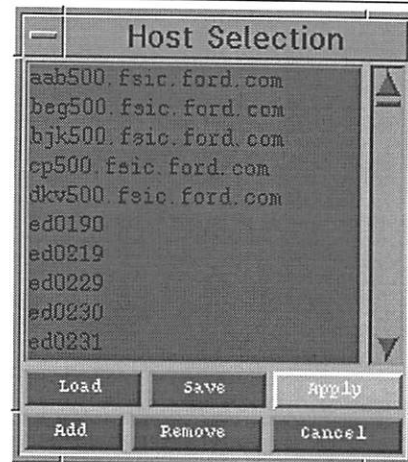


Figure 10: Host selection dialog

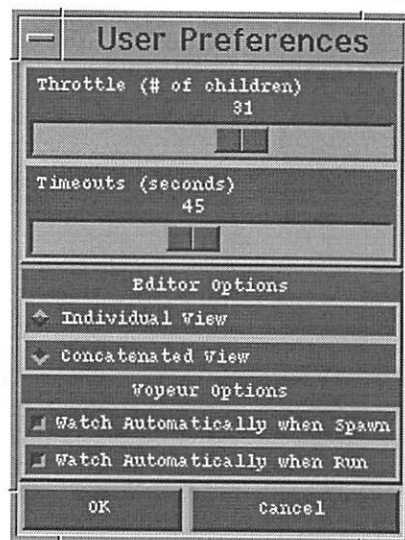


Figure 11: User preferences screen

Now we're ready to contact the hosts. Clicking on the “Run” button will cause Igor to contact the remote hosts, and end them the script to be run. A progress indicator [Figure 5] lets the operator know how many hosts are untouched, being-worked-on, or have completed the commands. When the sliders indicate everything is done (or even before that) the operator an click on “Refresh” and that will cause the RE's to be run against the results obtained. The results are shown in Figure 6. Some hosts passed OK (they already have the resolver in the file) some did not. The hosts which pass can be selected with the mouse, and then the “forget” button pressed.

This will drop these hosts from the host list. We do this, because only the hosts that need work are left in the list.

The remaining hosts are left onscreen. We can then load in a new set of commands [Figure 7], and a new set of regular expressions [Figure 8]. These will actually add the new resolver into the resolv.conf file. Pressing "Run" will cause the progress indicators to reappear, and when Igor is all done, we can see which hosts were modified successfully, and which were not.

If problems appear during the run, there's quite a few things that can be done to diagnose what happened. To actually look at the data retrieved from the hosts by selecting the hosts we're interested in, and then clicking "View Selected". The raw data retrieved from the remote host is shown in a vi session. A sample of the retrieved data is shown in Figure 9. The initial information shows the connection being established, the standard output and standard error are shown, separately. From this information you might be able to tell what's wrong on the host.

If more diagnostics (or repairs) are needed to individual hosts, the operator can double-clicks on a host in the list. An Xterm will open (running "rsh host") so that he can check things out manually. If a large number of hosts failed, the operator can rewrite the script and try it again.

Cautions

If Perl is the "Swiss Army Chainsaw" of UNIX, then Igor is a Gatling Gun loaded with Swiss Army Chainsaws – a useful tool or a terrible weapon

of destruction. Igor is the fastest way we know of to fix problems on our 700 hosts – it's also the fastest way to cause them. It should probably not be used by anyone who doesn't understand how it works. For example, if you were to mistype a Igor script to do `"/bin/rm -rf /tmp/*"` and had typed `"/bin/rm -rf /tmp /"` then ALL of your systems would be instantly erased. This is a normal pitfall for system administrators, it's only magnified with Igor.

The thought of implementing Operator safety features to Igor has entered our minds ("Are you sure?" type questions, etc...) and then swiftly left. One of the most powerful features of Igor is the fact that it doesn't get in your way. Once the structure of the commands are learned (there are only 4) and you've collected enough post-processing templates Igor just lets you do whatever is necessary...quickly. No fuss, no muss.

Author Information

Clinton Pierce is a System Administrator for Decision Consultants, Inc. and is currently assigned to Ford Motor Company. He is currently involved with integrating Solaris, AIX and IRIX workstations into a common look-and-feel environment. In addition, he teaches UNIX and Perl to consultants for DCI. Clinton can be reached via e-mail at cpiercel@ford.com, or U.S. Mail at Ford Systems Integration Center, 1000 Republic Drive, Suite 600, Allen Park MI 48101.

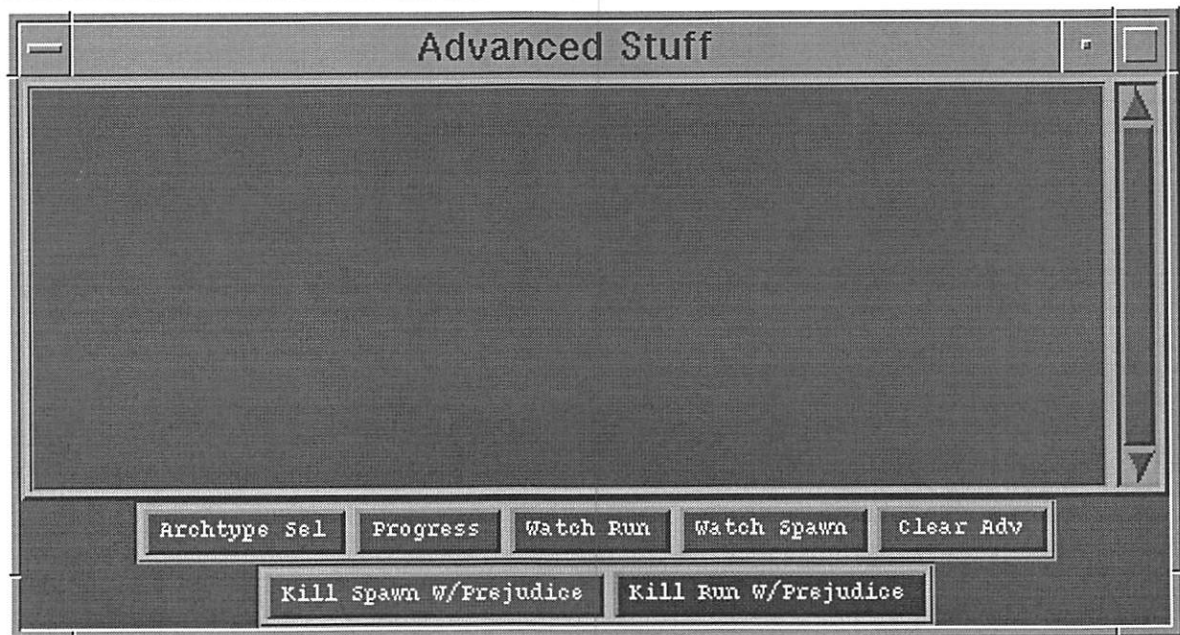


Figure 12: More buttons

Bibliography

- [1] Welch, Brent 1995 Practical Programming in Tcl and Tk Prentice Hall, Englewood Cliffs, NJ.
- [2] Wall, Larry, and Schwartz, Randal L. 1991 Programming Perl O'Reilly & Associates, Sebastopol, CA.
- [3] Lombardi, Christine, and Desimone, Salvatore 1993 "Systcl," Proceedings of the 1993 USENIX LISA Conference pp. 133, Monterey, CA.
- [4] Stevens, Richard W. 1992 Advanced Programming in the UNIX Environment. Addison-Wesley, Reading, Mass.
- [5] AIX DSMIT Guide and Reference Version 2.2 1994, Pub. Number SC23-2667

Centralized Administration of Distributed Firewalls

Mark Miller & Joe Morris – Bell Atlantic

ABSTRACT

Allowing the connection of various networks to one's internal network while keeping the internal network protected from unauthorized access is key to success in this information age. Firewall systems are currently the best known way of achieving this objective. Management of these firewalls in a secure, consistent and reliable manner is a strategy that is imperative to the successful interconnection of diverse networks. This paper describes the technical implementation of a paradigm enabling the administration of distributed firewall systems from a centralized location.

Justification

In November of 1995 the Business Continuity group was formed at Bell Atlantic, and charged with providing security for all systems connected to the Bell Atlantic internal network. One key responsibility was to provide firewall support, including support for existing firewalls, as well as the development of standards for the support of any new firewalls.

Prior to the formation of the Business Continuity group, a total of eight people in four different groups spread throughout the seven state region of Bell Atlantic performed maintenance of existing firewalls. System administration policies were inconsistent, and management felt a certain lack of control over the state of our connections to other networks. In response, a standard firewall architecture was developed using a combination of vendor supported software, public domain software and programs developed in-house. Currently, only three people in the Business Continuity group are needed to provide support for all existing firewalls, with less than five percent of their time dedicated to ongoing firewall management.

This change was seen as a very positive step by all parties involved. From an executive perspective, there was a reduction in the number of resources dedicated to performing firewall support. Those previously performing the function were generally elated as this was never meant to be their full time job, and this move enabled them to focus on their primary responsibilities. From a security perspective, it was advantageous because consistent repeatable processes were in place for building and managing the firewalls. This was a major step that allowed our security stance to go from reactive to proactive.

Defining the Architecture

Bell Atlantic has an internal TCP/IP based network consisting of 25 class B address spaces assigned based on geographic locations and connected in a hierarchically manner. This internal network, known as

BAInet, is connected to several other networks, including, the Internet, other Bell Atlantic networks, and a few customer sites. The manner in which these interconnections are constructed ranges from simple filtered routers to full-fledged DMZ lans with bastion hosts. The services provided through these interconnections range from standard Internet services to application specific communication channels.

To aid in the ease of administration, a standard firewall architecture was adopted. This standard defines the guidelines to be used in the design of new firewalls. The key principles used in defining the architecture are:

- All firewalls are designed to eliminate any Single Points of Failure.
- All management functions are performed centrally.
- No management functions are done in the clear over the network.
- Firewall rules are built on the philosophy that "whatever is not expressly permitted is forbidden."

This architecture also specified hardware and software standards which were to be used.

Selecting A Standard Hardware Architecture

Firewall systems can contain both filtered routers and firewall computers (generally Unix). In developing a standard firewall configuration, the first decision was that of the hardware platforms to be used. Sun was chosen as our firewall computer vendor and Cisco was chosen as our router vendor.

Sun was chosen for the following reasons:

- There is a high level of familiarity with the hardware and company.
- Tools are, in general, most readily available on Suns. Stated another way, tools are generally not unavailable for Sun.
- Bell Atlantic has a pre-existing support contract with Sun.

- The scalability of Sun's hardware platforms enables the selection of correctly sized system for individual firewalls.
- Existing Unix support groups in Bell Atlantic are most experienced with Suns. This enables us to pass certain support functions to this group and exploit economies of scale.

The firewall servers are designed to be redundant, and to eliminate single points of failure. For example, duplicate network interfaces are used, and mirrored internal disks are configured to allow booting from either disk. High availability is greatly enhanced by these measures as well as the use of Uninterruptible Power Supplies at all sites.

Cisco was chosen as our router vendor for the following reasons:

- Administrative access can be provided in a secured manner.
- There is pre-existing expertise in writing Cisco rules.
- Cisco is the de facto Internet standard.
- Firewall/1 allows creation of Cisco rules.

One concern voiced over the decision of using only one router vendor is the argument that, "if a weakness is found in the operating system of the router, then all firewall routers from that vendor can be subverted." This problem was avoided by always including a firewall server sandwiched between two routers. This design necessitates that both the filtered router and a firewall be compromised to gain unauthorized access. Also, possible confusion caused by using multiple vendors' filtering languages is avoided.

Securing the Operating System

The decision to use Sun as our hardware base lead us to use Solaris as our operating system. A decision was made to use Solaris 2.5 based primarily on the fact that it includes most security patches to Solaris 2.4, and it has numerous performance enhancements over Solaris 2.4. Also, by using the latest version of Solaris, we avoid the common vendor retort of "load the latest patch" when confronted with problems.

Once the operating system was selected, a process was developed for hardening the operating system. The process consisted of:

- Loading the base operating system.
- Removing unnecessary or dangerous programs (e.g., sendmail, ftpd, telnet).
- Securing permissions on key files and directories (e.g., removing setuid and non-owner write permissions).
- Installing various software packages (e.g., ssh, tiger, tripwire, FDDI package).

The entire process was scripted and documented so it could be performed in less than an hour. The philosophy used in the design of the secure operating system installation process was threefold in nature:

- Prevent and detect unauthorized access to the firewall.
- Prevent and detect unauthorized root access of the firewall.
- Prevent further unauthorized access via a compromised firewall.

Installing the Firewall Software

The firewall product which was selected was Checkpoint's Firewall/1. The reasons for using Firewall/1 were:

- Firewall/1 is a highly respected firewall product.
- Firewall/1 rules are built using a GUI. Rules could be discussed with appropriate management, without requiring management to possess a high degree of knowledge in a filter rule language.
- Remote centralized administration functionality is inherent in its design. This function is not supported by many firewalls.
- We inherited it from existing firewalls. We had paid for the licenses, and it was deemed most cost-effective to use the pre-existing licenses.

There are also several features of Firewall/1 which are promising to us for future applications. First, support for SecurID authentication is built into the Firewall/1 proxies. This was promising to us as we already have a large installed base of SecureID card users. The inclusion of NAT and VPN functionality is also encouraging to us for future use.

Installing Monitoring Software

Monitoring of the firewall systems is performed by several means. Firewall/1's built in logging facilities are used for monitoring the activity of the firewall filters. Custom developed software is used for the monitoring of system resources. Finally, several public domain utilities are used for monitoring security related aspects of the system. Procedures have also been established and implemented for both regular review of the logs by a person, and automated analysis and reporting of error messages via scripts.

Firewall/1 allows the reporting of critical conditions by several methods including writing to a log file, popping up an X Window alert box, or specifying a program to run. We opted to run a custom program; specifically Firewall/1 is configured to use the syslog facility to enable real time message forwarding. Messages are generated by Firewall/1 whenever an inappropriate packet is detected.

System resources are monitored using a modified suite of scripts developed at Bell Atlantic for monitoring Unix systems. These scripts monitor memory, disk, and processor resources and send messages when certain thresholds are surpassed. All scripts are written in Bourne shell and report error or threshold conditions via the syslog facility.

Two public domain security programs are also used to monitor the system. We chose Tiger to perform regular audits of the system. Tiger is a tool from TAMU which verifies that permissions and ownerships of files on the system are not overly permissive. Tiger was chosen over several other available packages due to its ease of use and configuration. Any error messages which are generated are compared to a file containing expected messages, and any differences are forwarded to the central collection system for analysis.

The other public domain security tool we use is Tripwire from Purdue University. When initialized, Tripwire creates a database of permissions, ownerships, modification times and digital signatures of key files on the system. Upon subsequent runs, Tripwire will report any changes made to the attributes of those files. After Tripwire has been run for the first time, the database is copied onto a floppy diskette. The diskette is then made read-only, and mounted on the system, thus preventing tampering with the initial database.

One difficult selling point was the use of publicly available tools on the firewalls. The tools we decided to use are ones that have been reviewed and discussed, in depth, in the Internet community. They also have the benefit of being free. As Brent Chapman and Elizabeth Zwicky point out,

"Some people feel uncomfortable using software that's freely available on the Internet, particularly for security-critical applications. We feel that the advantages outweigh the disadvantages. You may not have the "guarantees" offered by vendors, but you have the ability to inspect the source code and to share information with the large community that helps to maintain the software. In practice, vendors come and go, but the community endures."

A modified version of syslogd is used to transport messages from our firewalls to our logging system. The modifications restrict syslogd to receiving messages only from the local system, i.e., it will not accept syslog messages from other systems. Syslog was chosen as our transport mechanism for several reasons:

- Syslog is easy to configure.
- Other alternatives for message transport are equally or less trustworthy (sendmail, snmp).
- Syslog is standard on all versions of Unix which we were planning on using.
- Commercial monitoring packages are more expensive and an "unknown quantity". The source code for syslog is freely available and reviewable.

Syslog is configured on the firewall to write the message to a log file, display the message on the console, print the message on a directly connected printer, and forward the message to a central syslog message collector.

The central error message collector receives the syslog messages and processes them using a software system called BASEMM (Bell Atlantic Server Error Message Monitor). BASEMM is an application designed to parse system-generated error messages. In essence it functions as a wrapper around the syslog log file (/var/adm/messages). Because of its design it will also parse any application message that uses the system "logging vehicle": syslogd. In general terms, BASEMM refers to the collection of applications that collect and report on data parsed by the central application, the BASEMM daemon (BASEMMD).

The various components of BASEMM are written mostly in Perl 5. Other than the various programs and scripts on the BASEMM client machines the only other BASEMM application not written in Perl is the one that does alpha-numeric paging, which is written in expect.

BASEMM parses incoming messages and deals with error conditions in a variety of methods. If the message is deemed critical, the message is logged, and optionally, key individuals are paged. If the message is an unusual message, i.e., one which has not been seen before, it is written to a log file for later review. If the message is a well known message, and not deemed critical to the continued operation of the system, the message is dropped. This data reduction capability is one of the most powerful functions of BASEMM. As an example, on one day BASEMM received 17,000 messages and reported on 120 of them. Without the data reduction capability of BASEMM the firewall managers would be swamped with data.

Regular review of log file is performed on a 24x7 basis by the Network Management group. This group has been provided with procedures for reviewing the log files and looking for anomalous messages. In addition, throughout the business day, a summary of unreconcilable messages is emailed to appropriate Business Continuity individuals.

Defining System Administration Processes

All traditional system administration functions on the firewalls are performed by the Business Continuity district except those operations requiring physical access to the systems not physically co-located with members of the district. For those remote systems, there is a limited set of individuals (site administrators) at the remote site which have physical access to the firewall. Physical security is strongly enforced at all firewall locations, requiring, at a minimum, a card-key to enable auditing of physical access.

The root password is controlled and only known to the Business Continuity district. Any administration requiring root authority is the responsibility of Business Continuity. Root logins are only permitted from the console. For remote sites, the root password is sealed in a tamper-evident container, and stored in a safe located in a physically secured area of the

building. This is to allow the site administrator to have root access in an emergency situation in which no one from the Business Continuity district is available. Examples of this may include, a power outage, or an emergency change to the firewall rules. Once this password has been compromised, a new root password is generated and secured.

The ssh tool is installed on all firewall systems. Ssh is a replacement for the set of Berkeley r-commands, with the enhancement that all sessions can be encrypted, and no passwords are sent in the clear over the network. Ssh is really the key to enabling remote administration of the firewall systems. The use of ssh combined with token authentication also enables secure system administration to be performed from home. Ssh also allows for use of "one-time" passwords and because all traffic is encrypted greatly reduces the chance of a compromise. All methods of remote access are removed from the firewall machines leaving ssh as the sole method of remote access.

Regular scans using several IP Scanners and intrusion attempt tools are also performed. Bellcore's Pingware is used to scan for general Unix security holes. Netcat (or nc) is used to scan servers protected by the firewall to assure they only offer appropriate services. IP Watcher from EnGarde monitors specific connections and traffic patterns when required.

A change management process has also been put into place to assure proper notification of planned changes. There are two classes of changes which can be made to the firewall, confidential and non-confidential. Confidential changes include security related changes, and any firewall rule changes. Confidential changes are communicated in a secure manner and require approval from both the Business Continuity director, and the director of the client organization. Non-confidential changes include general hardware changes and changes which affect connectivity to BAlnet. Non-confidential changes are scheduled via the Bell Atlantic standard change control process, and require approval from only the director of Business Continuity. The Bell Atlantic standard change control process requires review of a change by a change control board. After a change has been completed a standard checklist is used to verify that the proper levels of functionality and security are in place.

Summary

Our experience has shown that distributed firewalls can be securely and effectively managed by a centrally located organization. It is hoped that other individuals can benefit from our experience.

Material discussed in this paper can be obtained at ftp.bell-atl.com/BC. There are ASCII, postscript and MS Word versions of the entire paper and a copy of the slides in Powerpoint format at this site. The paper can also be viewed in HTML format at <http://www.bell-atl.com/BC>.

Acknowledgments

Thanks to John Conner, Steve Geria, Bill Hill, Bill Irwin, Mark Krause, Harry Leung, Ming Ma, and Brad Passwaters for providing suggestions which greatly enhanced this paper.

Author Information

Mark Miller is currently employed at Bell Atlantic in Silver Spring, MD as a Distinguished Member Technical Staff. He has worked for six years in the areas of Unix system administration, Unix system programming and Internet security. While at Bell Atlantic, he was awarded the Presidents Award for Outstanding Service for work performed to enhance network survivability. He was co-creator of the first Bell Atlantic Web server. He has done graduate study at James Madison University, and received his bachelors degree in Mathematics and Computer Science from Eastern Mennonite University. Reach him via US Mail at 13101 Columbia Pike, Room 208H, Silver Spring, MD 20904, USA. His electronic mail address is [<marcus@bell-atl.com>](mailto:marcus@bell-atl.com).

Joe W. Morris retired from Federal service after 20 years. While assigned to the National Security Agency, he was a senior computer security officer, senior system administrator. At the agency, he won a Scientific Achievement Award in 1991, and the prestigious Directors' Productivity Improvement Award the following year. Both awards were for network firewall solutions. He has won numerous other DoD awards as well. Since joining Bell Atlantic in January 1994, Joe has introduced a number of technical innovations. He was co-creator of the first Bell Atlantic Web server. He created the Bell Atlantic Server Error Message Monitor (BASEMM, used to monitor major projects) and the NOVA Client Outage Notification Service (NCONS), and also automated metric creation from the NCONS database. Joe is currently assigned to firewall and other security technology projects. Reach him via US Mail at 13101 Columbia Pike, Room 209B, Silver Spring, MD 20904, USA. His electronic mail address is [<joe@bell-atl.com>](mailto:joe@bell-atl.com).

Bibliography

- Centerpoint Software. Firewall-1 2.0 documentation. Available with the Firewall-1 product.
- D. Brent Chapman and Elizabeth D. Zwicky. *Internet Firewalls and Network Security*. O'reilly & Associates, Sebastopol, CA 12995.
- William R Cheswick and Steven M. Bellovin. *Firewalls and Internet Security, Repelling the Wily Hacker*. Addison-Wesley Publishing Company, Reading, MA, 1994.
- R. Christopher E. Hare and Karanjit Siyan. *Internet Firewalls and Network Security*. New Riders Publishing, Indianapolis, IN 46290.
- Dan Farmer and Eugene H. Spafford. "The COPS security checker system". In *USENIX*

Conference Proceedings pages 165-170, Anaheim, CA, Summer 1990. *A package to audit systems for vulnerabilities. The paper is available for anonymous ftp from ftp.cs.purdue.edu as /pub/spaf/security/COPS.PS.Z.*

Gene Kim and Eugene H. Spafford. *The design and implementation of Tripwire: A file system integrity checker*. Technical Report CSD-TR-93-071, Purdue University, 1993. *A package to audit systems for vulnerabilities and evidence of hacking attacks. This paper is available for anonymous ftp from ftp.cs.purdue.edu as /pub/spaf/security/Tripwire.PS.Z.*

Joe Morris. "Implementing Active Data Network Security at Bell Atlantic". In *Computer, Network and Information Security Symposium Proceedings*, Baltimore, MD, June 1996.

Joe Morris. "The Bell Atlantic Trusted Network Segment: A Firewall Solution". *Practicum for System Analysis and Design*, Potomac College, Washington, D.C., June 1996.

David R. Safford, Douglas Lee Schales, and David K. Hess. "The TAMU security package: An ongoing response to Internet intruders in an academic environment". In *Proceedings of the Forth Usenix UNIX Security Symposium*, pages 91-118, Santa Clara, CA, October 1993. *A detailed look at a hackers activities in a university environment – and what they did to stop them. The paper is available for ftp as part of the TAMU security package (Tiger).*

Wieste Venema. "TCP WRAPPER: Network monitoring, access control and booby traps". In *Proceedings of the Third UNIX Security Symposium*, pages 85-92, Baltimore, MD, September 1992. Available for ftp from ftp.win.tue.nl at /pub/security/tcp_wrapper.ps.Z.

Tatu Ylonen. *Secure Shell*. A package that replaces the Berkeley "r commands" (rsh, rlogin) with cryptologic enhancements. Available from <http://www.cs.hut.fi/ssh>.

Shuse: Multi-Host Account Administration

Henry Spencer – SP Systems

ABSTRACT

At the beginning of 1995, Sheridan College urgently needed an organized way of administering a large number of user accounts spread across multiple Unix systems. With 6000+ accounts on a network that had recently undergone dramatic and ill-coordinated growth, the situation was already nearly unmanageable; with the user population forecast to double in autumn, disaster loomed. NIS served reasonably well for the simple task of distributing password files, but maintaining the master copy was proving problematic, creating directories and configuration files for new users was a very ad-hoc process, and there was no obvious place to record assorted supplementary information.

The response was to create a new software package, dubbed “Shuse” for “Sheridan user management”. A central daemon maintains the user database, which is in a fully extensible text-based format. Rather than use a commercial database package, the daemon simply keeps the entire database in its (virtual) memory, and the master copy on disk is optimized for rapid updates rather than efficient access. (RAM is cheaper than database packages nowadays.) Update requests go to the central daemon; it invokes auxiliary processes on other hosts as necessary to create, destroy, and move user files.

Shuse is written essentially entirely in Expect, an extended variant of Tcl. Inter-host communication is done by using Expect’s process-control primitives to fire up *telnet* processes; bulk data transfer is done via NFS. About 100 lines of C code, in three small auxiliary programs, provide services that are not present in Expect. A not-accidental byproduct of this approach is near-automatic portability and correct functioning even in a heterogeneous network.

Shuse is in operational use, currently administering over 20,000 user accounts (the forecasts were low). Various problems were encountered along the way, some easily solved and some requiring considerable unforeseen effort. The use of Expect has been a clear success, performance problems were easily resolved, and the central-daemon approach has worked well.

The Problem

Sheridan College¹ is a large community college, with several campuses located in the outer suburbs of Toronto. Its computing facilities are centered on a set of DEC Alphas running DEC UNIX (formerly named OSF/1), although there are also large numbers of PCs, a scattering of high-end graphics machines for the animation courses, and a wide variety of odds and ends (everything from 486/Pentium boxes running BSD/OS to one or two moldering VMS machines). The facilities have expanded enormously in the last few years, and there has been only limited advance planning on how to deal with the rapid growth.

Sheridan has been moving steadily toward an accounts-for-everyone policy, exacerbating the usual difficulties of large numbers of accounts on multiple hosts. In early 1995 there were over 6000 users; this was forecast to double within the year. Many of the users have very little experience with computing,

especially shared multi-user computing, and the combination of heavy course-loads, non-technical backgrounds, and high turnover limits what can be done with user education. (Instructions like “change your password by doing an *rlogin* to server so-and-so and running *yppasswd*” are worse than useless.)

By the beginning of 1995, rapid growth and limited planning had made the situation almost intolerable. Funding shortages prevented major increases in staff, routine maintenance chores like creation of new accounts were absorbing large amounts of staff time (to the point where software problems were not getting solved because nobody had time), and continued growth threatened total collapse. Improvements were urgently needed, and in particular, really had to be in place before the September 1995 onslaught. At this point, I was brought in to Do Something About It.

Some potential difficulties were *not* present. All hosts do share a common password file, partly because they share file systems quite extensively via NFS. While there is a lot of heterogeneity “around the edges”, the major servers are all the same type of machine running the same operating system. (This

¹The one area where LISA attendees might perhaps have heard of Sheridan is that its computer-animation program has an international reputation.

didn't help as much as one might think, however, because it was not clear that this simple situation would persist.) The number of hosts involved as major participants is quite small: the primary problem was large numbers of users, not large numbers of computers. Finally, the network and the major servers have been reasonably reliable, and it was decided that there is no requirement to preserve full functionality in the presence of dead servers or partitioned networks.

Existing Software

An admittedly rather cursory look at existing solutions to this problem revealed little that seemed helpful.

Some system suppliers offer proprietary account-management software for networks of their machines, but Sheridan's network was already slightly heterogeneous and might easily become more so, so a system-specific approach was unattractive. The supplier packages also have an annoying habit of being menu-driven GUI-based interactive programs, which may be easy to use when creating a single account, but are severely unsuited to environments where 5000 accounts must be created in a week or two. Besides, DEC would be the only reasonable supplier for such a thing in this case, and DEC didn't appear to offer anything suitable.

The MIT Athena project has a Service Management System [1] which addresses this problem. (Indeed, it is somewhat similar to what we eventually built.) Unfortunately, it relies on a commercial database package *and* on other Athena software, and this didn't look like it was going to drop easily into Sheridan's existing environment.

At the time, we were not aware of GeNUAdmin [2] or AGUS [3], which might perhaps have been suitable.

Finally, one thing that was very clear was that Sheridan wanted a definitive solution to the problem, not a temporary bandage for the wound. This precluded various ad-hoc solutions which might have postponed the crisis at the cost of more effort later.

Design

Although some complications have been added to the original design, the basic elements have been fairly stable.

The fundamental approach was largely determined by consideration of one issue: coordinating updates. The orthodox way to do this, in a shared system, is some kind of locking protocol... but that presents problems in an NFS environment. None of the usual Unix file-based locking techniques work reliably with NFS's shoddy imitation of Unix file semantics. Reliable locking in such an environment requires using a supplementary protocol to consult a daemon somewhere; this is the approach taken by NFS's own

locking primitives, but unfortunately they are notoriously buggy.

Given that we were going to have to implement our own daemon anyway, the obvious approach was just to have it do all the work. Locking is unnecessary when all requests are funneled through a single "secretary" process. A dedicated central-information-server system was available to serve as the daemon's host, and its reliability and uptime were sufficiently good that the extra complications of distributed redundancy could be avoided. So we decided to implement a single central daemon, which would respond to queries, perform database updates, and invoke subordinates on other servers as necessary.

With this approach, synchronization is a non-issue, since only one process ever modifies the database. At least for starters, we decided to avoid reintroducing concurrency via threads: there is a single stream of control, operating in an event-based loop. The payoffs are a complete absence of locking overhead, fast updates, and vastly easier debugging (that last being particularly attractive in view of the hard deadline).

This approach was even more attractive because it permits a very useful optimization: the central daemon can be left running permanently, and can simply cache the entire database in its memory. One might think that this technique would be suitable only for small databases, but done well, it scales up quite effectively – memory is cheap. In any case, a brief analysis indicated that the amount of information for a particular user was unlikely to exceed a few hundred bytes, and this would require only a few megabytes for the expected user population. (More importantly, some quick tests showed that nothing dire would happen if this prediction was significantly exceeded.)

Obviously, it is still necessary to have an on-disk copy of the database, so that updates will survive both planned downtime and crashes. With all read-only accesses satisfied from the daemon's memory, the on-disk copy can be optimized for cheap and simple updates rather than rapid access to large amounts of data. After some quick feasibility testing, we decided to simply make each user's data a separate file in a simple text format. This does make the daemon's startup rather slow, since it has to read thousands of tiny files, but with a permanently-running daemon this is not needed often. Experiments indicated that on an otherwise-quiet system, reading 10-20,000 small files took only a few minutes, which seemed tolerable for a relatively rare event. We considered subdividing the files into a directory tree, but experiments indicated that just keeping them all in one directory was quite workable on a modern system.

For the file format itself, we briefly considered various extended versions of the classical *passwd*-file format, but decided against it. Any format with a fixed number of fields suffers when requirements

change, as witness all the creative things that have been done with the “GCOS” field in the *passwd* file. While it would be necessary to generate a *passwd* file from the database, we wanted the database itself to be flexible and extensible, so it could contain *all* the information about a user and would not need supplementing with auxiliary databases as new needs appeared. We did opt for a text-based format, partly just for simplicity, partly because this makes a wide variety of Unix tools useful for setting up the database or doing emergency surgery on it. A user’s database file looks something like Figure 1. The *server* field contains a code identifying the system the user’s home directory resides on; there is a separate control file which maps server codes to host names, to simplify changes in host configuration. The *schema* field contains a code indicating how to build an initial home directory for a new user; it is passed as an argument to the script that actually builds the directory. Fields like *office* and *workphone* contain information that is assembled into a suitable “GCOS” field for the *passwd* file; since we want the Shuse database to be the primary database, not a derived one, we store the information broken down by meaning, instead of a preformatted version appropriate to one specific version of the *passwd* file.

A single centralized daemon could not do the entire job. In particular, when creating or deleting users, it would be necessary to operate as *root* on the host holding the user’s home directory, and the limitations of NFS required that to be done locally. This was also necessary for a different reason: Sheridan mounts only subsets of its filesystems on its individual servers, so the central server host cannot necessarily see the filesystem which would have to be updated. It seemed that it would be necessary for the daemon to invoke an auxiliary program on the other, “slave”, servers. (Having this done as needed by the daemon, rather than at regular intervals by *cron*, would

propagate updates more quickly and avoid unnecessary overhead.)

We decided that the auxiliary program would read a description of what users *should* be on its host, then examine the host to find out which users were actually present, and then act to correct any discrepancies. This seemed likely to produce more robust operation than having the daemon send update commands to the auxiliary.

At this point we were starting to need names for the programs. We dubbed the whole system “Shuse” (pronounced like “shoes”), the daemon “shused”, and the slave-server auxiliary program “shusetie”.

The one other piece of machinery which had to be fitted in, somehow, was a user interface for talking to *shused*. The actual user interface was a somewhat secondary concern, especially since it seemed that there might have to be more than one, but we needed a way to talk to the daemon. To simplify implementation and separate the major concerns somewhat, we decided to have a separate “gatekeeper” program, *shusedgate*, invoked by *inetd* as required. The gatekeepers implement whatever authentication of credentials is appropriate, and then pass commands to the daemon and responses back, communicating with the daemon via a set of FIFOs. Apart from network interface and authentication, their role is to enforce timeouts on interaction and shield the daemon from possible interference by uncooperative users.

When network connections get involved, security is an obvious worry. The right way to deal with this in the long run, clearly, is with encryption. As a stopgap measure, since Sheridan already relied heavily on NFS being trustworthy², we decided that network

²Not necessarily a safe assumption, but that’s the way it was.

```

name      spencerh
passwd    76hgfu645fmvt
passwd@   806860944 (Thu Jul 27 12:02:24 1995) spencerh
uid        8172
gid        15
home       /home/apollo/it/spencerh
shell      /bin/sh
server     it
schema     n
status     active
status@    807477421 (Thu Aug 3 15:17:01 1995) root
fullname   Henry Spencer
workphone  1-416-555-4444
office     E108
mailname   henry.spencer
changed    807988782 (Wed Aug 9 13:19:42 1995) root

```

Figure 1: User database entry

connections would pass only “hey, wake up and look at this” requests, with all crucial information being passed via the file system.

Although the initial user community was to be mostly the system administrators, a fairly flexible permission scheme was clearly desirable. At one extreme, users had to be able to change their own passwords (we briefly considered using more traditional mechanisms for that, but decided that having Shuse handle *everything* was simpler than dividing the responsibility). At the other extreme, the system administrators had to be able to make fairly arbitrary changes. And there are a variety of interesting levels in between, such as help-desk personnel, who should be able to interrogate the database and do some limited operations like changing passwords but should not be permitted to do more drastic alterations.

To provide a flexible permission scheme, we tag each daemon operation with a “category”, and a control file specifies which categories of operations are open to which users. At one extreme, a few read-only operations are in category “harmless” and are available to everyone. At the other, arbitrary editing operations and the ability to shut down the daemon are in category “overlord”, which is restricted to a small set of users calling only from the central server machine itself.

Implementation Approach

With the design outlined, implementation started. The main constraint on it was that Shuse simply *had* to be functioning for the September 1995 student intake.

With some trepidation, we decided that Shuse would be written essentially entirely in Tcl [4,5]. Experiments suggested that performance would be adequate, and the use of a very-high-level language looked like it would speed up development considerably. Crucial portions could always be re-coded in C if necessary. This basic approach was quite successful in earlier projects [6], and Tcl seemed a better choice than the Unix shell (which tends to be slow and clumsy unless the task at hand is suited to the Unix utilities) or Perl (which tends to be ugly and unmaintainable).

We quickly settled on Expect [7,8], one of the most popular Tcl extensions, rather than “raw” Tcl. Initially, this was done because we anticipated uses for Expect’s ability to start and control other processes. It turned out that Expect also has a number of small amenities which make it a more complete programming environment than Tcl, which was envisioned as a minimal extension language rather than an independent programming language. (For example, Expect can catch Unix signals.)

We used Expect’s special I/O primitives in fairly minor ways (although see the discussion of *shuselace* later). The one area where we used Expect’s facilities

more seriously was in calls to slave servers, to invoke *shusetie*. The actual invocation of *shusetie* was done by *inetd* on the slave server, but the daemon did have to be able to make a call across the network. Rather than add primitives for this, or adopt one of the existing Tcl networking extensions³, we used Expect’s primitives to invoke *telnet*, specifying a non-standard port number to reach *shusetie* instead of *telnetd*. This may sound a little ugly, but in fact it is quite simple and practical, and has the bonus that portability is almost automatic: all the system-specific complications of networking are invisible.

To make Shuse easier to maintain, each of its programs reads in a configuration file as part of startup. Rather than parsing the configuration file and interpreting its contents, the program simply “sources” it, running it as part of the program’s own Tcl source. This makes it possible for the configuration file to contain not only the obvious variable settings – permissions, pathnames, etc. – but also Tcl procedures. As a case in point, the procedure that builds a *passwd* line from a Shuse database entry resides in *shused*’s configuration file, so that arbitrary changes in *passwd* format can be accommodated without diving into the main sources.

Gritty Details

Certain aspects of the implementation posed unexpected difficulties. We anticipated some of these, while others came as unpleasant surprises. (In some cases, we accepted marginally-satisfactory early solutions simply to get Shuse functioning in time; not all of the issues mentioned here were fully resolved for September.)

We had originally envisioned that extracts from the database, such as the *passwd* file or the user lists for *shusetie*, would simply be assembled by *shused* as necessary. It turns out that digging through the whole database every time such a thing is needed is relatively costly. This means that (for example) building a *passwd* file is expensive, and looking up a user by student number or mailbox name is very slow. Moreover, the information rarely changes much, so rediscovering it each time is wasteful.

Shused now builds internal auxiliary databases at startup, and updates them accordingly when relevant information changes. In some cases, the more costly updates are postponed until *shused* appears to be idle. For example, after startup *shused* uses idle time to build a copy of each user’s *passwd* line, and keeps those copies around. This permits pumping out a complete copy of the *passwd* file in a few seconds, when needed.

The single-threaded design of the daemon is awkward when long-running chores have to be done,

³Tcl itself acquired networking primitives in release 7.5, early in 1996, but that was about a year late for Shuse.

because an interactive request should not be delayed arbitrarily waiting for such a chore to finish. Long-running chores must either be broken up into small pieces, so that interactive requests need not wait too long, or be farmed out to auxiliary processes to take them out of the critical path entirely.

A particular problem area is that an update of a slave server can be very slow. The bigger slave servers contain thousands of home directories, and merely enumerating them all for comparison with a user list is a slow operation when user load is heavy. The biggest performance problem in early Shuse operations was long delays in interactive requests when *shused* was waiting for responses from *shusetie* running on a slow slave. When the systems were busy, the extremes of the response time were utterly unacceptable.

We briefly considered multi-threading *shused*, but apart from certain practical problems – it's not something Tcl does well – it seemed unnecessarily general for what was, after all, a somewhat specialized problem. We tackled this one from the other end: during startup, *shused* spins off a “flunky” process, dubbed *shuselace*⁴, which does all calls to the slave servers. The flunky makes calls to *shused* using (almost) the standard user command interface, with minor special privileges. (Expect's I/O primitives make it trivial for *shused* to listen for input from two sources instead of one.) *Shused* itself maintains a queue of work to be done by the flunky, and provides “user” interfaces which do things like removing one item from the queue. The flunky uses a *shused* command to pick up a work item to be done (e.g., “update server *nova*”), goes away and does it (taking as long as necessary), and then uses another *shused* command to report success or failure. The first version of the flunky was mostly code transplanted intact from the innards of *shused*, and setting it up took only a day or two's work. It was entirely successful, and response time has never again been a significant problem.

During development of Shuse, we were generally preoccupied with the daemon and its auxiliaries, and did not give much attention to the user interface. We obviously needed some sort of command interface to test the daemon, so a simple program that sends the daemon a single command gradually appeared, more as a debugging tool than a finished user interface. Naturally enough, it was fairly promptly pressed into service as a user interface. While it is somewhat inefficient – for bulk operations, one would prefer to be able to send the daemon more than one command at a time – it works sufficiently well that there has been little incentive to replace it. In particular, it is exactly what is wanted for writing scripts.

⁴In retrospect, we should probably have named the slave-server program *shuselace* and the central-server flunky *shusetie*, since the flunky manipulates the slave-server programs rather than vice versa, but it's too late now.

The one additional user interface that had to be provided was a naive-user password changer. We re-implemented the *passwd* command (and *yppasswd* as well) as an Expect script that requests old and new passwords, does appropriate checks⁵, and then calls *shused* to make the change. (Naturally, *shused* itself also does some checks before permitting the change!) This required adding another auxiliary C program, 30 lines of code which invokes the password-encryption routines and outputs the result.

The idea of making slave-server updates idempotent, by having *shusetie* compare existing users against a list of users who should be there, was a good one. It turned out to be a bit harder to implement than we expected. For one thing, it's purely and simply difficult to enumerate all the home directories on a server unless the server's directory structures are laid out to make this easy. For another, the comparison approach handles additions and deletions relatively easily, but can't be gracefully extended to handle moving or renaming users. We ended up doing substantial revisions to the structure of both *shuselace* and *shusetie* to implement a more general command facility within them, so *shused* could order specific operations done. Moreover, this involves some relatively fancy footwork to ensure that such operations are not lost if one of the servers crashes at an inopportune moment, and also some slightly more sophisticated authentication to assure *shusetie* that the thing sending commands is really *shuselace*.

One particular problem in the implementation of *shusetie* was disk quotas. The so-called user interface of the quota system is a disgrace to Unix: inflexible, interactive only, and completely lacking in reasonable primitives for system administration. To cap it off, DEC reinvented the wheel here: when they implemented a new filesystem type, instead of extending the existing quota commands to handle it, they added a new set in parallel – with the same crippling deficiencies in functionality, and some unhelpful changes in data format – so that on a mixed system, you may need to edit a user's disk quotas twice, with two different commands, to get them all!

Fortunately, Expect came to the rescue here. Since the quota facility does at least let you choose which editor you want to use to edit the quota data, we originally thought we'd just have it invoke *ed*, which we could drive with an Expect script. In the end, it turned out to be simpler to move some of the intelligence into the editor, so *shusetie* now manipulates the environment to make the quota commands invoke a little customized editor written in Expect. There is still the annoyance of having to do this twice, via two different sets of quota commands, but some careful

⁵We note that it is *vastly* easier to change or improve the *is-this-a-good-password* test when the program is written in a very-high-level interpretive language!

design of the editing primitives made it possible to do this fairly painlessly.

Successes

The bottom line is: it works. We're still discovering things that need improvement, but the September 1995 crisis was averted, and as of May 1996 the system was managing over 20,000 user accounts. (The predictions turned out to be low – instead of doubling, the user base more than tripled.) Response time is good since the implementation of *shuselace*, and the staff workload for routine administrative chores is declining.

The central-daemon approach is a weak point in theory but it seems to be adequate in practice. Our opinion is that unless unusual requirements are present, it's better to put effort into making a central server reliable than into making the software do without one. Compared to a more distributed approach, a single central daemon enormously simplifies debugging, synchronization, and management.

Using Expect was a big win. We couldn't possibly have met the schedule using C or the equivalent; in fact, we barely met it using Expect. Very few of the problems were a consequence of the interpretive language, and many of the rapid and simple solutions were a consequence of it.

Although Tcl, and hence Expect, is extensible, we did not find it necessary to do this. The option to add language extensions written in C always existed, but in practice we found that the few missing primitives were more easily implemented as separate programs, invoked as needed. For example, the gatekeeper invokes a 30-line C program, which does a *getpeername()* and a *gethostbyaddr()* and prints the result: the name of the host an incoming call is from.

The in-memory-database approach works well. We did end up adding some more RAM to the central server. (We note with some annoyance that conventional system interfaces are too ready to page out seemingly-idle process memory, and don't provide a way to say "let this process make as much of a pig of itself as it wants, and don't page it out unless you really must".) The response time for simple database queries is entirely dominated by the communications arrangements.

The update performance of the file-per-user on-disk database has been excellent, and although it takes several minutes for the daemon to start up and read in all those files, this is a minor nuisance rather than a serious problem. We have occasionally contemplated implementing facilities to dump out the daemon's in-memory database in some form that would permit rapid reloading, given that most daemon restarts are planned, but to date it hasn't been worth the trouble.

The extensible text-based format of the database entries themselves has permitted a number of

unplanned additions and amendments. There will surely be more.

While interest in sophisticated user interfaces, e.g., for the help desk, remains, the simple send-one-command interface has been amazingly successful. In particular, an extensive body of scripts has grown up to reflect local policy and frequently-run database operations. We very strongly believe that we made the right decision: do the command-line interface first, leave the fancy graphics for later.

Problems

Not everything went smoothly. Apart from the implementation difficulties mentioned earlier, some broader issues deserve mention.

As one might predict, the customer wishlist changed and grew once an initial system was operating. Things that weren't even mentioned in the original specifications turned out to be major issues that needed substantial reworking of the software. For example, the original design included a very simple facility for automatically executing commands at specific times, vaguely modelled on the Unix *at* command, and this saw such extensive use that some major re-engineering work was needed to make it more practical and efficient.

The original design had little ad-hoc protocols for each communications path. Only the protocol used between the gatekeeper and the daemon was fully fleshed out and pinned down. Since then, many of the paths which originally needed very little sophistication have grown to need the full nine yards; for example, *shusetie* now provides a full command interface to *shuselace*. One thrust of recent work has been to encapsulate the gatekeeper-daemon protocol in a library module, and convert everything to use it; this is almost complete, and has been a definite success.

Telnet connections, while adequate for commands, are suboptimal for bulk data transfer. Early versions of *shused* operations which returned very large amounts of data had mysterious problems with little bits of data loss. Debugging this was difficult, but we eventually established that the problem was in *telnet*, not in Shuse – it would seem that we were overstressing something in DEC's *telnet* or *pty* implementation. As a workaround, the few operations which routinely need to transfer large amounts of data were revised to do the transfer via the file system. The exact cause of the problem was never fully determined, and in fact we suspect that a system upgrade somewhere along the way may have fixed it. The new protocol library checks the length of data transferred in all operations, as a precaution.

We're also interested in the possibility of reimplementing some of the Shuse *telnet* communications paths using Tcl's new networking primitives. While this is of no real importance for Shuse's internal communications, improving the user-interface response

time would be nice, and it looks like most of the time spent there is in setup and teardown overhead rather than actual communication.

One area that has not yet been fully sorted out is logging and trouble reporting. After some unsatisfactory early experiences with coordinating multiple log files, the protocol library and some other facilities were extended slightly to let everything send log entries to the main daemon. This has helped, but we still need to do some more work in the area; it's particularly difficult to get satisfactory reporting in cases where final execution of an operation has to be delayed, e.g., because a server is down. Queueing up the work until it can be done is only half the job. It would be useful to have *shused* (or supporting software) maintain a current-status report on each slave server, to make ongoing problems more visible.

As mentioned earlier, response-time constraints and the single-threaded nature of the daemon require that time-consuming internal operations be broken up into smaller pieces. This has gotten easier as experience has accumulated, but that experience really needs to be distilled into a set of library routines that would make it relatively painless. There are a few infrequent operations which would benefit from being split up, but which are still in one piece because it's too much trouble.

The current implementation of Shuse is very much organized around running a single database: the users. In practice, this has been adequate for Sheridan's needs, and extensions in this area have been low priority. For example, Sheridan makes relatively limited use of Unix groups, so the group database is still managed manually. This would be less satisfactory for an installation which made more sophisticated use of group memberships.

In retrospect, the exact split of responsibilities between the software contractor and the Sheridan staff was not quite right. In particular, *shusetie* invokes external shell scripts to do things like creating or deleting user home directories. This works, but it would work better if the scripts were better integrated; in particular, error diagnosis would be improved. The lack of integration is an artifact of the responsibility split, and was largely forced on us by time constraints, but it's still a blemish.

Conclusion

Despite being written in an interpretive language and having to fire up other programs for, e.g., network communication, Shuse works fine and does the job. Performance has not been a problem since minor design errors were corrected, new functionality is easily added, and the system is coping well with databases of 20,000+ users.

Sheridan is happy, and commercial marketing of the software is being explored.

Acknowledgements

Although I wrote almost all of the Shuse software, a number of other people were involved in various ways. Cheri Weaver, then head of Sheridan's system-administration group, got me into all this :-)) in the first place. John Barber, her successor, has happily funded ongoing work and enhancements. Simon Galton handled the Sheridan side of Shuse, during its initial transition into production use, with skill and no small amount of bravery ("you're going on vacation *WHEN*, Henry?!?"). Seela Balkissoon, Rob Naccarato, Trevor Stott, and the other members of Sheridan's CSG have patiently used, commented on, and complained about Shuse while it was struggling towards operational maturity.

Availability

The Shuse software belongs to Sheridan College. Times are hard for educational institutions in Ontario, and there is local commercial interest in Shuse, so at this time it is not available free.

Author Information

Henry Spencer is a freelance software engineer and author. His degrees are from University of Saskatchewan and University of Toronto. He is the author of several freely-redistributable software packages, notably the original public-domain *getopt*, the redistributable regular-expression library, and the *awf* text formatter, and is co-author of C News. He is currently immersed in the complexities of implementing POSIX regular expressions. He can be reached as henry@zoo.toronto.edu.

References

- [1] Mark A. Rosenstein, Daniel E. Geer, & Peter J. Levine, *The Athena Service Management System*, in Proceedings of the Usenix Technical Conference, Winter 1988 (Dallas), Usenix Association 1988.
- [2] Dr. Magnus Harlander, *Central System Administration in a Heterogeneous Unix Environment: GeNUAdmin*, in Proceedings of the Eighth Systems Administration Conference (LISA 94, San Diego), Usenix Association 1994.
- [3] Paul Riddle, Paul Danckaert, & Matt Metaferia, *AGUS: An Automatic Multi-Platform Account Generation System*, in Proceedings of the Ninth Systems Administration Conference (LISA 95, Monterey), Usenix Association 1995.
- [4] John K. Ousterhout, *Tcl: An Embeddable Command Language*, in Proceedings of the Usenix Technical Conference, Winter 1990 (Washington), Usenix Association 1990.
- [5] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley 1994.

- [6] Geoff Collyer & Henry Spencer, *News Need Not Be Slow*, in Proceedings of the Usenix Technical Conference, Winter 1987 (Washington), Usenix Association 1987.
- [7] Don Libes, *Expect: Curing Those Uncontrollable Fits of Interaction*, in Proceedings of the Usenix Technical Conference, Summer 1990 (Anaheim), Usenix Association 1990.
- [8] Don Libes, *Exploring Expect*, O'Reilly & Associates 1995.

The Design and Implementation of a Network Account Management System

J. Archer Harris – James Madison University
Gregory Gingerich – Bell Atlantic

ABSTRACT

In many heterogeneous UNIX networks, users require accounts with attributes that vary by machine. This creates a complex management job. This paper reports on a software system for creating and maintaining network accounts.

Commercial and publicly available tools are briefly examined. Disadvantages of commercial tools are found to be high expense, difficulty of modification, and uncertainty of availability on future platforms. NIS and NIS+ are found to lack sufficient flexibility. Kerberos is determined to be an authentication system, not a user management tool.

The Network Account Management System (NAMS) is described. It addresses two key requirements. First, that users be able to have a single login across machines on a network, creating the abstraction of a "network account." Second, that these accounts allow for attributes such as home directory, expiration date, and login shell to vary by machine.

The Need for an Account Management System

The Computer Science department at James Madison University (JMU) owns approximately fifty machines that run a variety of implementations of UNIX (SCO, Sun, Ultrix, AIX, LINUX, etc.). There are approximately 300 users of these machines. The machines are spread over a number of labs, with various collections of machines serving different functions. One collection of machines might provide support for a geographic imaging class, while another collection might support a class in systems programming. Thus the set of users requiring access to a machine differs greatly from machine to machine. The set of users also changes from semester to semester. This leads to a complex management task.

Without using an account management system, a system administrator must set up accounts individually on each machine. If three hundred persons need access to fifty machines, this would require the entry of 15,000 accounts. Because it is often desirable to give users varying access levels and attributes on different machines, simply copying information from one system to another is not sufficient. Clearly it is also neither efficient nor feasible to individually maintain account records spread across fifty systems.

The primary goal of this project was to find or create a system that would allow for the creation of a single "network account" for each user. However, account information would not have to be identical for all machines on the network. For any piece of account information, one or more "clusters" of machines could be defined which would share information. Such clusters would have the ability to be independent and overlapping.

Consider the following example for users 'alpha' and 'beta'; machines **APE**, **BOA**, **CAT**, **DOG**, and **ELK**; and two pieces of account information, passwords and login shells. User 'alpha' may wish to have one password for machines **APE**, and **CAT** and another for machines **DOG** and **ELK**. They also may want one login shell for machine **APE** and another for machines **CAT**, **DOG** and **ELK**. The user 'beta' might also require password and login shell clusters, but they might be quite different from user 'alpha'. Figure 1 summarizes clustering in this example.

ACCOUNT	ITEM	CLUSTER
alpha	password	APE, CAT
alpha	password	DOG, ELK
alpha	login shell	APE
alpha	login shell	CAT, DOG, ELK
beta	password	APE, BOA, CAT
beta	login shell	APE, BOA
beta	login shell	CAT, DOG, ELK

Figure 1: Cluster summary

We sought a system which created the abstraction of a network account, but which had the flexibility to allow different attributes to be associated with various systems on the network. We wanted the account administrator to be able to control all accounts from a single workstation.

An Evaluation of Commonly Used Tools

Part of this project included researching network account management tools currently in use in UNIX environments. A search of the literature revealed that there are two types of systems available for heterogeneous UNIX system management. First, there is software commonly distributed with the UNIX operating

system or publicly available via Internet. This includes NIS, NIS+, and Kerberos. The other option is commercial products such as Tivoli System's TME, Xchange by Z/Max and CA-Unicenter by Computer Associates.

Literature research was augmented by discussions with several system administrators (two from the MITRE corporation, two at Bell Atlantic, and one at JMU), and various Internet resources such as Usenet postings, FTP sites, and the World Wide Web. These sources all seemed to suggest that commercial software is not often used. Reasons given for excluding commercial software included: (1) high expense, (2) reliance on a "closed" solution with no source code available for modification, and (3) no guarantees the software will work with future systems.

NIS, an Information Distribution System

The first system evaluated was the Network Information System (NIS), created by Sun Microsystems. This system, also known as yellow pages (yp), is widely used in UNIX environments today.

NIS is a service originally intended to allow the construction of "distributed computing systems" that are consistent in nature [Stern-92]. It does this by providing a "distributed database system" for common configuration files. Configuration files typically distributed by NIS include the UNIX hosts table, network services table, password file, and many others. By distributing configuration files from one central "server" to many "clients", NIS effectively allows multiple machines to behave as one.

One traditional use of NIS has been account administration via the */etc/passwd* and */etc/group* files. User accounts can be created on the server machine and distributed to all the clients. This approach effectively creates a network account of sorts, and works well when the account information needs to be consistent across all machines. One instance of this is when a number of identical machines have been purchased for collective use by a group. It is assumed that the users will want their accounts to have the same attributes, regardless of which machine they login to.

The environment within the Computer Science department at JMU is considerably different, as discussed earlier. It is desirable to be able to vary account information and create clusters of machines sharing information. NIS simply does not have this capability. Account information such as login name, password, and home directory must be shared either in its entirety or not at all. For example, it is not possible to distribute only an account's password and set expiration dates differently on each individual machine. (There is an exception mechanism built into NIS for the */etc/passwd* file. This allows each individual client to specify that accounts be added to, or removed from, the server's standard distribution. This process is cumbersome and does not provide partial sharing of an individual account's information.)

The inability to partially share account information makes it impossible to create independent and overlapping clusters under NIS. This was viewed as a major shortcoming and is due to the architecture of NIS. It is not likely simple modifications to NIS can overcome this problem. Because of this, it is difficult to justify the use of NIS in this project.

NIS+, the Successor to NIS

NIS+, released with the Solaris 2.0 version of UNIX by Sun, attempts to correct what many people see as drawbacks to NIS. Major advantages of NIS+ over NIS include secure data access, hierarchical network administration, and access to resources across domains [McManis-91]. However, a new problem has been created; NIS+ is not as widely available as NIS.

NIS+ improves security by adding authorization, authentication, and encryption not present in standard NIS. Secure RPC, also used by newer versions of the Network File System (NFS), provides for authentication and encryption during network communication. Secure RPC makes use of both DES and public key encryption [Stern-92]. This prevents sensitive (particularly password) information from being compromised through network sniffing and spoofing. Authorization has been addressed through the addition of access rights to each NIS+ database table. Read and Write permissions can be granted to the table's owner, group, and world. These access rights follow the standard UNIX filesystem model [McManis-91].

The ability to administer networks in a hierarchical fashion, and access resources across multiple domains is likely very useful in large corporate networks. However, the underlying structure of information distribution for NIS+ remains essentially the same as NIS. All account information must either be shared equally or not at all. Thus, NIS+ was also considered an inadequate solution to the problem.

Kerberos, an Authentication System

Kerberos is a product of Massachusetts Institute of Technology's Project Athena. It was designed with the goal of securing access to computers in an insecure network [Jaspan-93, Curry-90]. Kerberos authenticates users; it does not authorize them [Jaspan-93, Arnold-93, Sjogren-92]. There is a critical difference between these two activities. While Kerberos focuses on verifying that users and machines are who they say they are, this project was designed to centralize control of users' access privileges (authorizations) on the network.

Certainly authentication is an important network management issue, and can improve the security of account management systems. Future enhancements to this project will probably be concerned with authentication of machines on the network. There are several known mechanisms for authentication including Kerberos and RSA public key exchange. Authentication simply was not the issue this project was initially concerned with.

Tivoli's TME, an Enterprise Management Tool

Given the major problems of cost and lack of source code, commercial tools were not a primary focus of this project. We did take a closer look at TME [Tivoli-96], an enterprise management tool designed to allow centralized control of workstations, networks, and users with a common look and feel.

TME supports the central administration of workgroups with facilities for rapid addition or modification of the workstations and/or users associated with a workgroup. It appears well thought out, and covers most aspects of daily system administration. For our needs, it has two major weaknesses. First, it is not available on all the systems we are running. Second, it does not appear to support independent and overlapping information clusters. Although a more complete solution than NIS, it lacks the flexibility we sought.

Real Life System Management Experiences

As this project was not purely a theoretical problem, but also an attempt to create a practical solution to an everyday problem, it was deemed important to gather information "from the field." Discussions with system administrators at MITRE [Brown-95, Cwiek-94] and Bell Atlantic [Boss-95, Jordan-95] were very useful in providing some insights regarding commonly used techniques for account management. Both companies appear to rely on the use of NIS and NIS+. The administrators contacted were generally aware of the existence of Kerberos, and thought it was used "elsewhere" within their companies, but did not have any direct experience with it. It appears Kerberos may only be in use in applications considered critical or involving national security. None of these administrators had seriously considered 'enterprise' management systems such as TME and CA Unicenter. This may be due in part to the fact that these companies handle system administration primarily at the departmental level.

These discussions also revealed some interesting observations regarding the use of NIS and NIS+. NIS+ seems to work quite well for some groups at Bell Atlantic, particularly in networks consisting of nearly all Sun systems. An additional property of the Bell Atlantic networks is that a group of machines is typically assigned to a single project and users are assigned the same attributes on each machine belonging to the project. At MITRE, NIS+ was not used due to a more heterogeneous mixture of workstations (SGI, HP, IBM). NIS seemed a mediocre solution at best. While NIS worked well for distribution of some system configuration files, it failed at account administration. One reason for this failure was that users required different types of access to different machines. For one lab at MITRE, the situation became complex enough that the administrator used a "cut and paste" technique for accounts. Essentially, he was administering each machine's accounts independently.

This only worked because there were few machines involved (about ten), and a small number of users (about thirty). On a larger scale, this would quickly become a nightmare. For a second administrator at MITRE, the situation was a little different. NIS was used for some (but not all) machines. Exceptions were handled by making changes on individual machines. This meant that if a user in an NIS domain shouldn't receive access to a particular machine, that user was removed manually. This required logging into multiple machines to make changes to multiple configuration files. Again, this is a far from ideal situation.

One conclusion that can be drawn from these observations is that the usefulness of NIS and NIS+ for account administration depends on the makeup of the network and users. NIS and NIS+ both work best in an environment that: (1) consists of homogeneous workstations, and (2) does not require user access levels or configuration (i.e., home directories) to vary by machine. Conversely, these systems are not well suited to heterogeneous networks and varied user information. These are the conditions that have complicated account administration at JMU and they apparently are not unique.

NAMS, A New Approach

One conclusion that can be drawn from these observations is that the usefulness of NIS and NIS+ for account administration depends on the makeup of the network and users. NIS and NIS+ both work best in an environment that: (1) consists of homogeneous workstations, and (2) does not require user access levels or configuration (i.e., home directories) to vary by machine. Conversely, these systems are not well suited to heterogeneous networks and varied user information. These are the conditions that have complicated account administration at JMU and they apparently are not unique. The Network Account Management System (NAMS) is the result of an effort to design a better solution for account management. NAMS is an operational system currently responsible for distributing account information within JMU's Computer Science network.

Like NIS and NIS+, NAMS distributes information from a central server to individual machines. However, NAMS is a very specific system created for one purpose: to administer network accounts. If desired, NIS/NIS+ services can be used in conjunction with NAMS for distribution of other UNIX configuration files.

NAMS is considerably more flexible than NIS with respect to sharing of account information. It incorporates the notion of clusters, and maintains a database designed to store account information containing these clusters.

NAMS is a modular system, made up of programs which comprise the central database, network communications, and user interface. This modular

structure has allowed NAMS to be ported to new UNIX systems rather quickly, and should also allow its functionality to be easily expanded in the future.

Database Structure

Perhaps the most significant difference between NAMS and NIS is the nature of the data stored. NIS creates its database from the standard UNIX configuration files. It then distributes the information in the configuration files to all machines within its domain. NAMS was designed from the ground up with the idea of being an network account database as opposed to being a distribution system. The NAMS database contains the "items" of information considered relevant to user accounts on the network. The database structure is open in nature to allow the system administrator to customize it for his or her needs. A typical database would contain some standard information such as username, password, and home directory. However, NAMS is also capable of storing information like disk quota space, email addresses, email forwarding, and network directories owned by the user. These items are defined by creating a file named *items.data* in the NAMS data directory. Figure 2 below is a current listing of this definition file for the Computer Science department.

The three columns define: the information item's "name", the access permissions associated with that item, and a description of the item. The access permissions determine who can read and modify each item. Write access (w) also confers read access (r) to an item. The first letter indicates read and write permissions for the account owner. The second letter indicates read and write permissions for all other users in

the world.

In the example above, users are only permitted to change their passwords. To also allow users the ability to change their login shell, email, and forwarding addresses, one would need to change the rights field in the "shell", "email", and "forward" lines from "rr" to "wr".

While not shown above, NAMS also maintains a list of system administrator accounts that have read and write access to all items in the database.

Clusters

Information stored in the NAMS database has the ability to be grouped into clusters. For example, a single account might have two passwords defined on the network. Password A could be used for common, unsecured machines. Password B might apply to more sensitive machines. When either of these passwords is changed on one machine, all the other machines in that cluster will receive the updated information.

An asterisk at the end of an item name indicates the entry defines a machine cluster. In the above example, **passwd**, **home**, and **status** are clustered information items. Each account may partition the machines in the network into one or more clusters which share that particular information item. The other items are defined as global items, the data associated with that item is common to all machines on the network.

Figure 3 shows typical account entry in the NAMS database.

```

#
#      :ItemName:Access:Description
#
:uid      :rr:numerical user id
:gid      :rr:numerical group id
:realname :rr:real name
:deptid   :r-:department id number
:passwd*  :r-:encrypted password clusters
:passwd   :w-:encrypted password
:home*    :rr:home directory clusters
:home     :rr:home directory
:shell    :rr:login shell
:comment  :r-:comment
:email    :rr:JMU email address
:forward  :rr:email forwarding address
:status*  :r-:account status/expiration clusters
:status   :r-:account status/expiration date
:create   :r-:date account created
:expire   :r-:overall account status/expiration
:quota    :r-:disk space quota
:dirlist  :r-:list of owned directories

```

Figure 2: A Listing of the Item Definition File

In the example above, the first column of data is the account name. The second column holds the name of the information being stored and corresponds to the items of information defined in the *items.data* file described earlier. A starred item in the second field indicates this line defines the systems in a cluster. The third column stores the "cluster number". A cluster number of '0' is used for non-clustered information such as a user's real name. Each defined cluster occupies two lines: one line lists the machines in the cluster, the other specifies the value for the cluster. For example, the following two lines would define a password cluster for machines named **ruff**, **sys10**, and **tch3**, and the encrypted password used by those machines.

```
:gingergl :passwd* :1:ruff,sys10,tch3
:gingergl :passwd :1:encrypted_entry_A
```

A second cluster is used to define the password for all remaining systems.

```
:gingergl :passwd* :2:*
:gingergl :passwd :2:encrypted_entry_B
```

Note the use of the '*' above to indicate all machines not otherwise defined. This entry is required in the database for each clustered item to ensure that no machine will have an undefined value for an item. Another example of clustering in the above entry is the "status" field. The following two lines define a group of machines to which the user has access through 12/95.

```
:gingergl :status* :1:sys1,sys2,tch3
:gingergl :status :1:12/95
```

A second cluster defines systems this user does not have access to.

```
:gingergl :status* :2:thor,mutt
:gingergl :status :2:noaccess
```

The third cluster defines an expiration date of 05/95 for all remaining systems.

```
:gingergl :status* :3:*
:gingergl :status :3:05/95
```

Access could be denied to systems in any of these clusters by changing the expiration date to a value of "noaccess".

Information Distribution

In order to distribute information NAMS relies on the use of TCP/IP network communications [Comer-91, Hunt-92, Stevens-90]. There are currently two types of information flow between machines running NAMS. The first type consists of updates to the central database located on the server. The second type consists of updates sent from the server to individual clients.

Server Updates

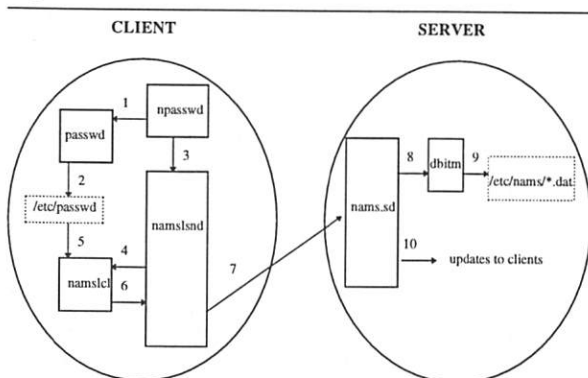
When an item of information (such as an account password) has been changed on a client machine, the server can also be updated with the change using the program **namslsnd** (short for NAMS Local SeND). This program retrieves information stored locally using a subprogram (shell script) named **namslcl** (NAMS LoCaL information). It then connects with a daemon running on the server called **namssd** (NAMS Server Daemon) and passes it the updated information

```
:gingergl :gid :0:500
:gingergl :realname:0:gregory gingerich
:gingergl :deptid :0:csmajor
:gingergl :passwd* :1:ruff,sys10,tch3
:gingergl :passwd :1:encrypted_entry_A
:gingergl :passwd* :2:*
:gingergl :passwd :2:encrypted_entry_B
:gingergl :home* :1:*
:gingergl :home :1:/homes/gingerg
:gingergl :shell :0:/bin/ksh
:gingergl :email :0:ginger@vax1.jmu.edu
:gingergl :forward :0:micro_greg@jmu.edu
:gingergl :status* :1:sys1,sys2,tch3
:gingergl :status* :2:thor mutt
:gingergl :status* :3:*
:gingergl :status :1:12/95
:gingergl :status :2:noaccess
:gingergl :status :3:05/95
:gingergl :create :0:01/92
:gingergl :expire :0:12/95
:gingergl :quota :0:30
:gingergl :dirlist :0:/homes/gingerg
```

Figure 3: An Account Entry in the NAMS Database

via a UNIX socket. The server daemon places the updated information in the NAMS database using a program called **dbitm** (DataBase ITeM) and sends a completion code back to the client. After terminating the connection with the client, the server makes a list of all other clients in the updated item's information cluster. It then broadcasts the update to those systems, as described below.

Perhaps the item most frequently updated is an account password, which is accomplished using the shell script **npasswd**. The steps involved in changing one's password are described below and illustrated in Figure 4.



0. Execute the NAMS program **npasswd**.
1. **npasswd** executes **passwd**.
2. **passwd** obtains a new password and modifies **/etc/passwd**.
3. **npasswd** executes **namslsnd**.
4. **namslsnd** executes **namslcl**.
5. **namslcl** extracts account information from **/etc/passwd**.
6. **namslcl** returns account information to **namslsnd**.
7. **namslsnd** sends the account information to **namslcl**.
8. **namslcl** passes the account information to **dbitm**.
9. **dbitm** modifies the NAMS database.
10. **namslcl** broadcasts the update to all other machines in the same password cluster.

Figure 4: Update password

Client Updates

There are two mechanisms used for updating clients with information stored on the server. The first involves the client contacting the server and requesting an update. The second method has the server broadcasting messages to client machines listening continuously for updates.

The client may contact the server daemon to request an update using a program called **namsupc** (NAMS UPdate Client). The request is either for information on an individual account or for all accounts (which are valid on the client machine). The program **namsupc** establishes a UNIX socket

connection with server daemon **namssd** and sends the list of items it wishes to be sent. The server daemon then checks to see which machine is requesting the update using an IP number lookup. If the request is for all accounts, it constructs a list of the accounts that are valid on the machine making the request. The socket is then used by the server to send the information to **namsupc** which uses **namslcl** to store the information in the client's standard account files. The process is illustrated in Figure 5.

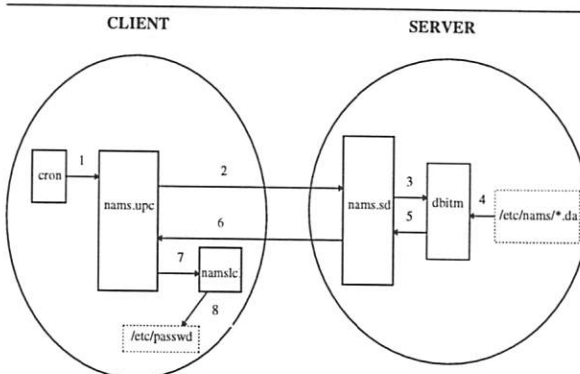


Figure 5: Update information on client

The second form of client update involves the use of the client daemon **namscd** constantly listening on a privileged UNIX socket for updates. When the server daemon receives an update, it broadcasts the update to **namscd** on all the clients in the same information cluster. The client daemon uses **namslcl** to store that information in the standard system account files. See Figure 6.

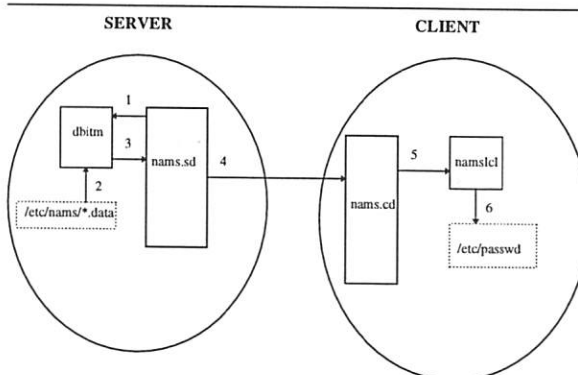


Figure 6: Update clients

The **namsupc** program is typically used at startup time to synchronize the client with the information stored on the server. It can also be scheduled by the system administrator to run (using a UNIX cron entry) several times daily. While this mechanism alone could provide updated information, it does not allow for changes to be distributed throughout the network quickly. The client daemon **namscd** provides a method for faster distribution of changes. This is particularly desirable in the case of password updates.

Security, A Growing Concern

When discussing security issues, there are two important rules. The first is "need to know." Users of a system should not be told more about underlying security than necessary [Arnold-93]. The second rule is that if you must rely on the first rule for security of your system, you are in real trouble. Keeping this in mind, we will try to explore the security issues related to NAMS at a high level of abstraction.

Security in NAMS

Weaknesses in the design or configuration system software can have drastic effects on UNIX security [Arnold-93, Garfinkel-91, Curry-90, Spafford-88]. Associated with a system like NAMS are a number of security concerns. Potential problem areas include NAMS programs disclosing or corrupting data in NAMS data files or in standard system files (like the */etc/passwd* file), sending information to unauthorized machines, and receiving bogus information from unauthorized machines.

All NAMS data is stored in a directory to which normal users need no access. All NAMS programs access NAMS data through the **dbitm** NAMS utility. By creating a special user id 'nams', setting permission so only user 'nams' (and 'root') may access the data, and running **dbitm** setuid to 'nams', access to NAMS data can be restricted.

The only other program requiring special privileges is the program which sends local client information to the server, **namslnd**. It must run setuid to 'root' so it can access standard accounting files and thus is a potential security loophole. For this reason, special care has been taken in its development. It is designed to refuse an account (other than root) access to another account's information.

The client and server daemons are not setuid programs and therefore are of no security concern when called directly by a user. However, they must be run by 'root' when started as daemons. All network communication uses privileged UNIX sockets to prevent non-'root' access. Additionally, limited authentication is performed based on the IP number of the client sending information.

Sniffing and Spoofing

Concern over "sniffing" and "spoofing" security attacks has risen dramatic in recent years. Sniffing is the monitoring all network traffic, searching for critical information such as passwords. Spoofing occurs when a rogue machine pretends to be another machine. If the machine being impersonated (spoofed) is trusted by other machines on the network, this can be exploited through protocols such as **rlogin** and early versions of NIS and NFS. The degree to which

sniffing and spoofing is a problem depends greatly on such factors as a network's physical security, the degree of access to the network from global networks, the type of users on the system, and the sensitivity of information stored on the systems. In our case, sniffing and spoofing were initially a secondary concern.

Given NAMS must send account information over the network, the initial version was susceptible to an attack by a sniffer. Although NAMS does not send plaintext passwords over the network, the first version does send encrypted passwords from one machine to another. On many networks, a sniffer could pick up these encrypted passwords and attempt to break them through the use of a program such as Crack. The first version of NAMS provides minimal protection against such attacks.

Spoofing is another method by which NAMS could be attacked. Clients running the daemon **nams.cd** could be updated with false information by a machine masquerading as the server. However, physically securing the server machine would make it relatively difficult to spoof. One also has the option of not running the client daemon and obtaining updates only with explicit requests via **namsupc**.

A more likely scenario is that a client would be spoofed. If users have physical access to client machines, it would be easy for them to turn off one of these machines. They could then plug a personal laptop into the network or replace the operating system on an existing machine. By assuming the IP number of the disabled client machine, the rogue system could fool the server into believing it was communicating with a legitimate client. Assuming access to the NAMS software or knowledge of its protocols, an update request from this bogus client could then change passwords or other information on the server.

To address these sniffing and spoofing concerns, Version 2 of NAMS protects network communications using independent one-time encryption keys. Based on a system described in [Rubin-96], each client is assigned a unique "client-key", known only to the client and the server. The client-key is the concatenation of three 8-byte DES keys. At the start of each session, the client sends the server a unique "session id". The session id is an integer between 1 and 32767, and changes with each communication between client and server. A pseudo-random function takes the "session id" and the client-key, and using three iterations of DES, generates a "session-key" using the formula shown in Figure 7. For the remainder of the session, all communications between client and server are encrypted with the session key. The ability to encrypt or decrypt with the session key authenticates the client or server system.

$$S\text{-Key} = \text{DES}(\text{DES}^{-1}(\text{DES}(S\text{-Id}, C\text{-Key1}), C\text{-Key2}), C\text{-Key3})$$

Figure 7: Session key encoding

A Security Flaw Discovered

One interesting flaw discovered during the writing of this paper is related to a design problem with user-definable clusters. If users were given permission to change both their password and their password clusters, they could defeat the security inherent in the clustering.

1. A user establishes two password clusters. One cluster of machines is considered more sensitive than the other.
2. The user's account is compromised on the 'low security' cluster, either through a weakness of the system or user carelessness.
3. A cracker who has gained access to one of the lower security machines takes the following action to gain access to one of the sensitive machines. First they change the account's password clustering to include the compromised machine in the "sensitive" cluster. Next they update the password on the compromised machine. The cracker now has access to any machine in the sensitive cluster.

This problem is particularly bothersome because compromise of one system's root account (allowed to update all accounts) could lead to compromise of an entire network of machines running NAMS. For this reason, NAMS now limits the ability to change clustering to system administrators on the NAMS master. Potential threats such as this must be dealt with as they are discovered.

Performance

NAMS has been in use in the Computer Science Department at James Madison University since the fall of 1995. Currently, NAMS is managing 294 accounts on a network of six Sparc Stations running SunOS and one Intel based PC running SCO ODT. In the past, the network managed by NAMS has also included five IBM Risc systems running AIX and thirty additional Intel based PCs running SCO ODT (reallocations of resources made those systems no longer available for UNIX use).

On a lightly loaded system, execution time for updating account information on a client machine averages around 2 seconds. Updating all 294 accounts takes around 130 seconds. No significant difference has been found in the speed necessary to update a PC client and a Sparc 20. Performance degrades if a number of clients request simultaneous updates of all their accounts. For four simultaneous updates of all accounts, execution time averaged 271 seconds.

It should be noted that this level of performance has been acceptable to date, since the update times only impact the speed with which changes to accounts occur. Validation of logins is nearly instantaneous, since each client machine still relies on its own */etc/passwd* and */etc/shadow* files to authenticate users. The current performance limits do create a

scalability issue. Performance of a network of several hundred or thousand machines is not known at this time. Options for enhancing performance would include using *ndbm* for the database engine, converting pieces of code from shell scripts to C/C++ code, and using a "Master/Slave" server configuration similar to NIS.

Summary, Future Enhancements to NAMS

This project has resulted in the successful development of a network account management system. The current implementation has successfully shown that the concept of clustering is both useful and viable. It has also raised some interesting security issues related to user-definable clusters.

NAMS was purposefully built in a modular fashion with the anticipation that it will grow over time. A menu-driven or window-based interface for the system administrator would also provide significant benefits. Currently, the administrator utilities consists primarily of line oriented add-account and remove-account commands. Beyond this, the system administrator must understand the database structure of NAMS and edit it manually or through use of the *dbitm* command. Also a utility is needed which includes a well organized structure for accessing and updating information, some intelligence to simplify the administrator's job and automate database changes, and possibly another layer of abstraction to hide complex details.

NAMS software is available via anonymous FTP at <ftp://ftp.cs.jmu.edu>.

Author Information

J. Archer Harris received his MS and Ph.D. degrees in Computer Science from SUNY at Stony Brook. He is currently an associate professor of Computer Science at James Madison University. He can be reached via U.S. Mail at Department of Computer Science; James Madison University; Harrisonburg, VA 22801. Reach him electronically at arch@cs.jmu.edu.

Greg Gingerich received a BS-CS from James Madison University in 1995. He is currently a programmer with the Broadband Systems group at Bell Atlantic working on a Data Network Management System. Reach him via U.S. Mail at Bell Atlantic Broadband Systems; 13100 Columbia Pike, POD A33; Silver Spring, MD 20904-5247. Reach him electronically at bftxddm@Bell-Atl.com.

References

- Arnold, N. Derek, *Unix Security: A Practical Tutorial*, McGraw, 1993.
- Boss, Todd, Bell Atlantic (Silver Spring, MD), Personal discussions, 1995.
- Brown, Philip, The MITRE Corporation (McLean, VA), Personal discussions, 1994.

- Comer, Douglas, *Internetworking with TCP/IP*, Prentice Hall, 1991.
- Curry, David, "Improving the Security of Your Unix System", Technical Report ITSTD-721-FR-90-21, SRI International, April 1990.
- Cwiek, Daniel, The MITRE Corporation (McLean, VA), Personal discussions, 1994.
- Garfinkel, Simson, and Gene Spafford, *Practical Unix Security*, O'Reilly and Associates, 1991.
- Hunt, Craig, *TCP/IP Network Administration*, O'Reilly and Associates, 1992.
- Jaspan, Barry, Kerberos Users' Frequently Asked Questions, OpenVision Technologies, November 1993.
- Jordan, Thomas, Bell Atlantic (Silver Spring, MD), Personal discussions, 1995.
- McManis, Chuck, and Saqib Jang, Network Information Service Plus (NIS+): A White Paper, Sun Microsystems, 1991.
- Rubin, Aviel, "Independent One-Time Passwords", *Computing Systems*, Vol.9, No.1, Winter 1996
- Sjogren, Sam, "A Bite Out of LAN Crime", *UNIX Review: Interoperability Supplement*, Spring 1992.
- Spafford, Eugene H, "The Internet Worm Program: An Analysis", Technical Report CSD-TR-823, Purdue University: Department of Computer Science, November 1988.
- Stern, Hal, *Managing NFS and NIS*, O'Reilly & Associates, 1992.
- Stevens, W. Richard, *UNIX Network Programming*, Prentice Hall, 1990.
- About Tivoli, "<http://www.tivoli.com/products/products.html>".

UNIX Host Administration in a Heterogeneous Distributed Computing Environment

Gregory S. Thomas, James O. Schroeder, Merrilee E. Orcutt, Desiree C. Johnson, Jeffrey T. Simmelink, & John P. Moore
– Pacific Northwest National Laboratory

ABSTRACT

Managing a computing environment consisting of hundreds of systems from multiple vendors and hundreds of users dispersed over several buildings of a research campus is a complex task. We developed a data model for representing pertinent information about our computing environment and implemented this model in a database that tracks equipment, users, and accounts. We created software that automates much of the data collection, maintains the database, and provides easy access to commonly requested information. For example, a user can find out which hosts he or she has an account on, or learn the hardware and software configuration of any host on the local network. We also created software that automates many of the tasks involved in account creation, deletion, and modification, which reduces the burden on support staff by allowing privileged users to manage accounts on their own systems.

Introduction

This paper describes a UNIX Host Administration (UHA) system that is being developed to help manage a scientific computing environment.

The Environmental Molecular Sciences Laboratory (EMSL) is a national collaborative research facility located at the Pacific Northwest National Laboratory (PNNL). The EMSL is currently under construction and is scheduled to become operational in late 1997, but its computing environment is already in production use by staff working in interim facilities.

The research mission supported by the EMSL computing environment is characterized by the frequent introduction of new hardware and operating software. The integration and stabilization of these systems is often a challenge. As the research community supported by EMSL computing continues to extend outward to include more international collaborators, careful management of accounts becomes critical.

There are two primary goals of the EMSL UHA system:

- Simplify common administrative tasks so they can be performed by clerical staff and/or users, thereby focusing limited system administration resources on complex integration issues.
- Improve the efficiency and integrity of processes that track users, accounts, and equipment, as needed to meet security requirements and good business practice standards.

Our approach to meeting these goals has been to implement a simple but effective system that includes a central data repository managed by a relational

database and a suite of tools that run on the target hosts and interact with the database to perform tracking and administration services. The simplicity of the system design and extensive use of perl [18] should minimize the effort required to port the code to clients running new operating system releases.

The focus of the initial implementation has been on automated data collection, data reporting, and account administration, but the infrastructure is now in place to rapidly deploy other functions as needed.

The tools implemented to date provide several high-payback benefits:

- Account administration procedures can be delegated to users and clerical staff.
- Account administration procedures are consistent across platforms.
- Policy decisions are codified.
- Tracking of users, accounts, and equipment is centralized.

The following sections of this paper discuss the computing environment, system architecture, database, account management software, related work, current status, and future directions.

Computing Environment

The EMSL's computing environment consists of a wide variety of computers and network equipment, ranging from parallel supercomputers to high performance workstations to Macs and PCs. There are roughly 840 computers on site, including 380 Macs, 260 PCs, 160 UNIX workstations, and 40 X terminals.

UNIX workstations from most of the major vendors are present. From each vendor, there are different models of computers and different versions of

operating systems. An effort to consolidate the UNIX environment (i.e., reduce the number of hardware and operating system variants) is underway, but the environment will remain heterogeneous.

Most of the UNIX workstations run AFS [15], which provides a shared file space. Login authentication on many workstations is provided by the Kerberos [9] component of AFS. The goal is to move the passwords for all non-privileged accounts to the Kerberos registry.

EMSL computer users include PNNL staff members associated with the EMSL, post doctoral researchers, other visiting scientists, and remote collaborators from the international scientific community. At this writing there are approximately 300 users of EMSL UNIX systems, evenly divided between PNNL and non-PNNL staff members. When the EMSL becomes operational, it is projected that there will be 300 PNNL users and 2000 non-PNNL users.

EMSL policy is to manage accounts on a per-computer basis in order to meet security requirements. The current UNIX users have about 500 accounts; there are 3400 instances of these accounts altogether.

Account administration in the EMSL is a continuous process, characterized by steady activity without the peaks typical of the university environment. It is important that new accounts are opened in a timely manner, and, for foreign national users, that accounts be created only on approved machines. Equally important is that all instances of a user's account(s) be disabled when the user's association with the EMSL concludes.

System Architecture

This section gives a brief overview of the system architecture for the suite of UNIX Host Administration (UHA) tools developed for the EMSL. A simplified view of the UHA system architecture is shown in Figure 1.

The UHA suite provides a semi-automated method for managing accounts on heterogeneous distributed UNIX hosts.

The UHA suite uses a central database as the controlling data structure for users and hosts, and a host-resident UNIX data structure (e.g., `/etc/passwd`) as the source of account data.

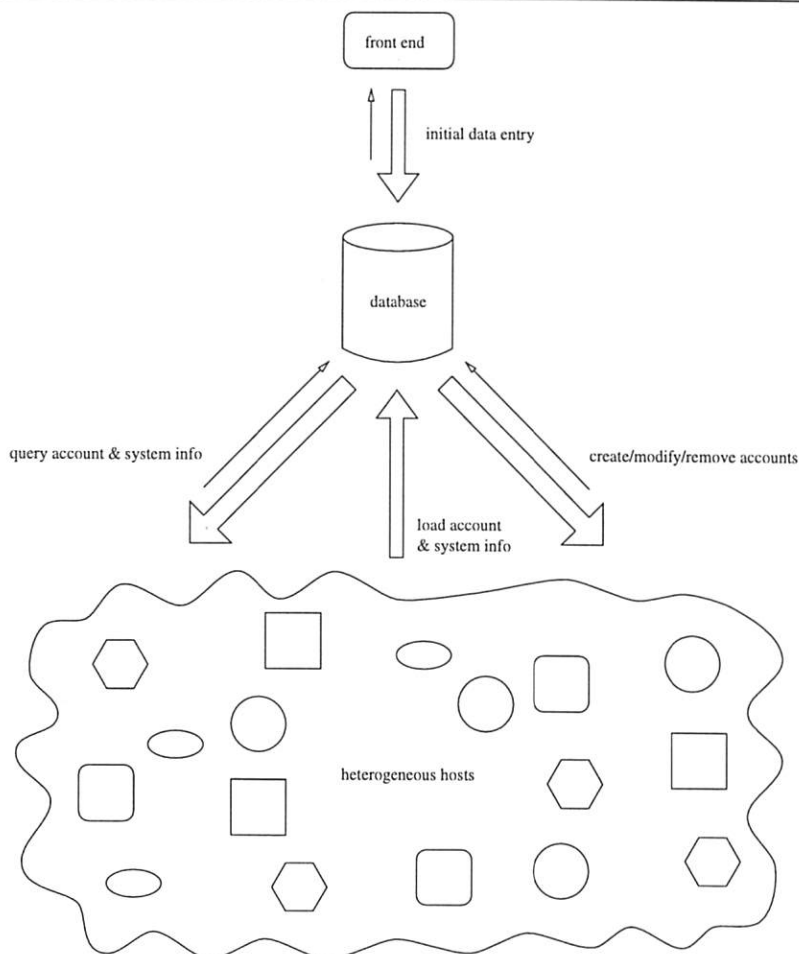


Figure 1: UHA System Architecture

The database is maintained by direct data entry through a spreadsheet interface, and by perl scripts run on distributed hosts to extract system and account information with which to update the database. Each time a new user is registered, a user profile is entered into the database.

Account information on distributed hosts is managed incrementally. The administrator or user invokes a script on the target host to create a new account. The script extracts user information from the database and executes architecture specific commands to add the account to the appropriate host-resident data structure (e.g., `/etc/passwd`).

Each host periodically uploads its system information (e.g., hardware model, operating system version, memory size) and information about active accounts into the database. Accurate system and account information for each host is thus maintained automatically, without manual updating of the database.

Commands are provided that allow users to query the database from the UNIX command line. Users can find out which accounts they own, which hosts the accounts reside on, what the hardware and software configuration of a host is, and so on.

Database

To help manage the EMSL computing environment, the Operations, Administration, and Support Information System (OASIS) database was created. The following sections describe OASIS: the data model, the software architecture, the implementation, server operation, writing data, and reading data.

Data Model

The OASIS data model contains approximately fifty tables. In this paper, the following four tables are of primary interest:

- **equipment**: hardware configuration, software configuration, owner, location, and inventory information for a computer
- **users**: name, identification information, location, and organizational affiliation for a person
- **user_accounts**: account name, account information (e.g., user identification (UID) number), and identification information for a person's UNIX account
- **account**: account information for an instance of a person's account on a particular host

A simplified version of an entity-relationship diagram for the subset of the OASIS schema corresponding to these four tables is shown in Figure 2. A single arrowhead denotes a one-to-one relationship, e.g., there is one **equipment** record associated with a given **account** record. A double arrowhead denotes a one-to-many relationship, e.g., there are many **account** records associated with a given **equipment** record.

There is one **equipment** record per computer, and one **users** record per person. Often there is only one **user_accounts** record per **users** record, but there can be more than one. Multiple **account** records may be associated with a single **user_accounts** record, one for each host on which the account exists.

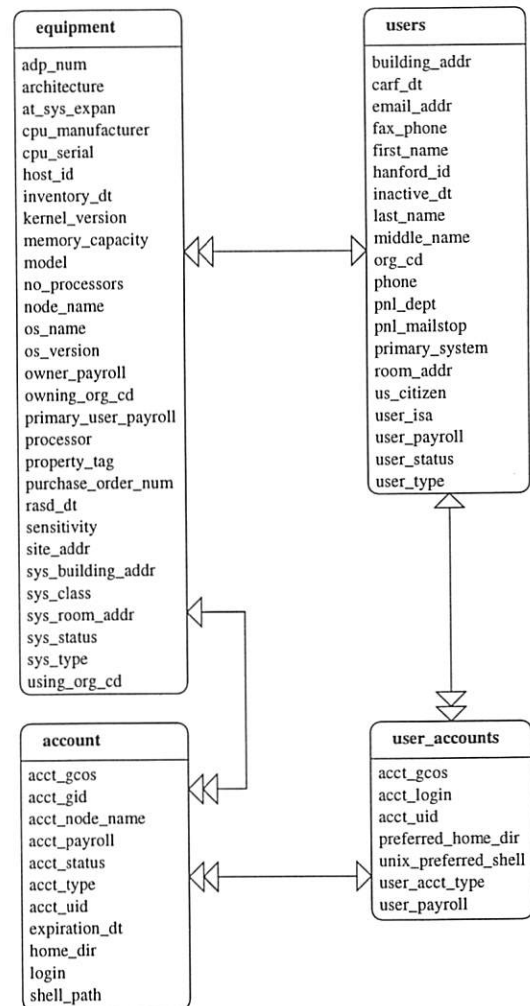


Figure 2: Partial OASIS Schema

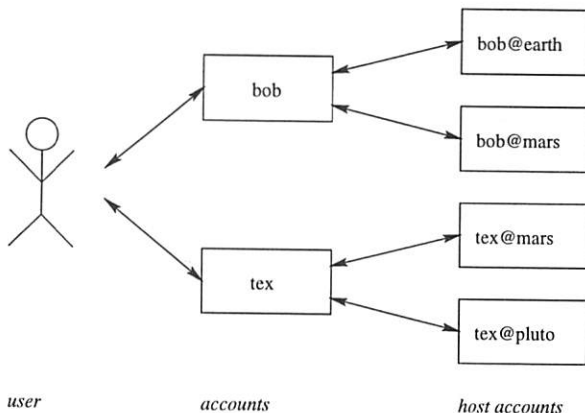
As an example, a user named Robert might have an account named **bob** for personal use and a different account named **tex** for doing TeX administration. If the **bob** account can access hosts **earth** and **mars** and the **tex** account can access hosts **mars** and **pluto**, there are two **account** records associated with the **bob** **user_accounts** record and two **account** records associated with the **tex** **user_accounts** record. Figure 3 shows graphically how records in OASIS tables model this situation.

Software Architecture

The database server runs on an EMSL workstation (the *database host*). Access to the database from other hosts is provided by the database vendor's client-server software libraries. The client libraries are

installed in the shared AFS file space for the other hosts to access. Most of the UHA software runs on the hosts and accesses the database via the client libraries. Some of the UHA software for database maintenance runs directly on the database host and uses an alternate interface (*isql* [13], the database vendor's command line SQL [14] utility) to the database.

Real World



Database Model

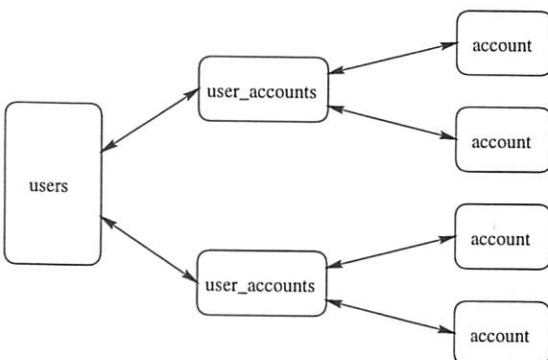


Figure 3: Account Representation in OASIS

A typical UHA application is written in perl. The application reads data from or writes data to the database via the sybperl [6] software package; sybperl provides extension modules that allow perl applications to call Sybase client-server library functions. This three-tiered layering of software is depicted in the client host boxes in Figure 4.

Most of the UHA applications do not invoke sybperl functions directly. Instead, they call UHA database library subroutines, which are part of a larger group of UHA library modules that perform tasks such as the following:

- Encode arrays into strings, decode strings back into arrays
- Compare, compute differences, search, merge, slice, sort, and perform other operations on one-dimensional, two-dimensional, and associative arrays
- Convert between byte prefixes

- Mimic the UNIX crypt program
- Authenticate jobs and users
- Parse flat files into database structures
- Transparently write messages to multiple destinations, e.g., a log file and the standard error output
- Extract host information
- Generate standardized log file entries
- Generate standardized input prompts
- Generate unique names for temporary files
- Manage file locks

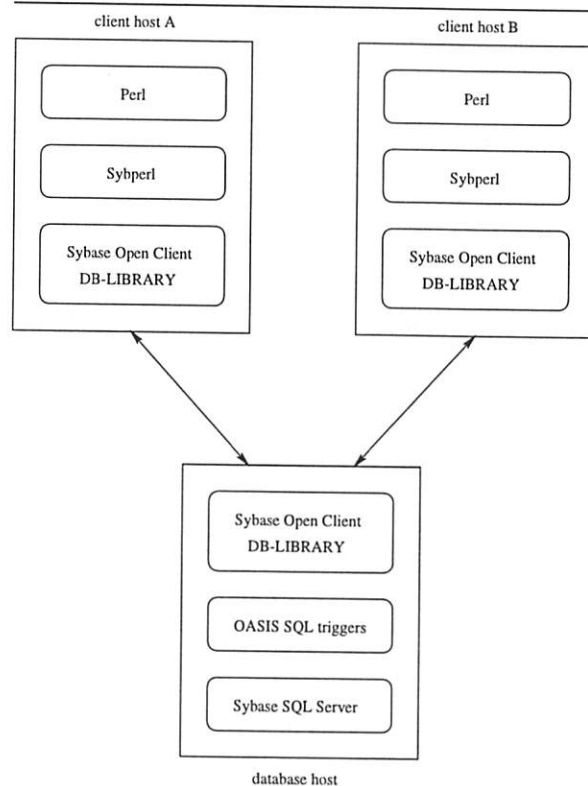


Figure 4: UHA Software Architecture

These perl modules facilitate code reuse across UHA applications and make it easier to extend existing applications and develop new ones.

The UHA library module that provides database access offers two avenues. The first allows access from local or remote hosts and employs sybperl and the Sybase client-server library functions. The second allows access only from the local host (the database host) and employs *isql*. Code implementing the first avenue is separated into three layers. The lowest layer is a wrapper around the sybperl functions (e.g., login to the database). The middle layer adds authentication and database connection opening/closing to support individual queries or sequences of queries (e.g., execute an SQL statement and return the result). The highest layer performs complete operations ranging from the simple (e.g., return the record matching a given key) to the complex (e.g., dynamically query the database for its field definitions and truncate candidate

data values to the lengths defined for the corresponding fields).

In addition to the UHA perl code, a separate body of SQL code is stored in the database itself. This SQL code implements the OASIS triggers and stored procedures. A stored procedure is a named collection of SQL statements stored in the database. A trigger is a type of stored procedure that is automatically invoked by the database when a command tries to modify a specified table or column. As shown in Figure 4, the OASIS triggers appear as a layer immediately above the database server.

Implementation

The OASIS database currently runs on a Sybase SQL Server 10.0.2 [12], hosted on a Sun SPARCstation 20 running Solaris 2.4.

Access to the database is provided by command line scripts and a spreadsheet front end. The scripts are written to run on perl 5.001m [17], syperl 2.0, and Sybase Open Client DB-Library 10.0.x [11].

A prototype graphical front end was written using a cross-platform Mac and UNIX tool, but has been discarded in favor of a (future) web-based solution. As an interim measure, Microsoft Excel is employed to provide a spreadsheet style interface (via Microsoft Query [5]) to the database that is familiar to clerical staff who maintain the user and equipment data. Use of this interface requires knowledge of the database structure, but it was chosen because it was easy to implement using readily available software.

Server Operation

A database server requires frequent attention. To reduce the operational burden, scripts were written to perform the following tasks:

- Back up the database
- Perform consistency checks on the database
- Start the server
- Stop the server
- Query the database for various information
- Compare records in the database against records in a file

Currently, the transaction logs of the database are dumped twice daily, a full database dump is performed nightly, and a suite of consistency checks is run nightly.

Automated filtering is performed on the output of the database back up and consistency check scripts. These scripts are normally run periodically via *cron*, which produces electronic mail messages containing output from the scripts. Rather than having a person scan several messages each day for problems, procmail [16] was used to implement a smart mail filter that screens normal messages and forwards only messages that indicate problems.

Writing Data

Many of the mechanisms for maintaining data integrity are incorporated into the database in the form of triggers and stored procedures. OASIS will not allow the deletion of a *users* record, for example, if that *users* record is still the owner of an *equipment* record, because one of the OASIS triggers forbids it. Another example is the stored procedure that generates a unique UID when a new user is registered in the database.

Implementing the integrity mechanisms once in the database saves the effort of reimplementing them for each different interface (e.g., perl scripts and a spreadsheet front end) and minimizes the potential for corruption by centralizing the integrity mechanisms. Furthermore, security is increased because the triggers cannot be bypassed; they can only be disabled by a database user who has owner or root privileges.

The following scripts load data into the database:

- *load_accounts* – upload information about the accounts on a host
- *load_sysinfo* – upload information about the hardware and software configuration of a host, using the *sysinfo* program [3] if it is available for the platform

They are run periodically on each host via a *cron*-based Automated Host Management System (AHMS), which is a PNNL-developed system complementary to UHA.

There is also a *bulk load* script that imports data from a file into the database, which is useful for initially loading tables with data from a spreadsheet or other database.

Reading Data

To avoid forcing users to learn a database query language (e.g., SQL) or the details of the database schema, the following scripts retrieve commonly requested data from the database:

- *show_acct* – list all instances of an account
- *show_profile* – list the parameters of an account
- *show_sysinfo* – list information about the hardware and software configuration of a host
- *show_user* – list demographic information about a user

Account Management Software

The following scripts simplify common account management tasks:

- *host_acct_create* – create a new account on a host
- *host_acct_modify* – modify an existing account on a host
- *host_acct_remove* – remove an account from a host

These scripts are used by system administrators and privileged users, and offer several benefits. Enabling privileged users to manage accounts on their

own systems is a win-win situation because it offloads work from the often overworked support staff and it eliminates the delay between when a privileged user (who is often the owner or primary user of a machine) wants an account change and when that change occurs. Having scripts that encapsulate the sequences of commands necessary to manage accounts is advantageous for several reasons:

- Administration procedures are consistent across platforms, no matter who performs them.
- Administration procedures are easier to perform because the scripts hide platform differences and execute procedures completely and correctly, thus saving time for experienced administrators who already know the underlying commands, and enabling novice administrators who are unfamiliar with the underlying commands to easily perform procedures they might otherwise struggle with.
- Numerous policy decisions are codified, e.g., foreign nationals must have an authorization record in the database before being granted an account on a particular host.

Before invoking a UHA account management script, a user must obtain root privileges, which is accomplished by running *sudo*. The *sudo* program, which is a modified version of the standard UNIX *su* program, uses access control lists to grant privileges based on user name, command name, and host name.

Related Work and Products

The UHA architecture is powerful but inherently simple. It is built on the concepts developed in similar packages and products. Several such systems have been described in the literature.

Moirra [8], from MIT Project Athena [2], is a service management system that feeds information from a central database to other Athena servers. Moirra is the architectural genesis for UHA and possibly other systems that use a centralized database as the definitive source of configuration and control data. However, redeployment of Moirra without other Athena services did not appear to be a pragmatic approach, and a full Athena implementation was more pervasive than desired.

Systems that perform comprehensive management of host configuration and accounts include GeNUAdmin [4] and lcfg [1]. These systems use information maintained in a central data repository to build control structures that are then loaded on the client. In contrast, UHA's account management commands are interactive and incremental, targeted at the primary user(s) of a host who uses *sudo* to execute the commands with root permission. UHA tracking functions work in the opposite direction, periodically collecting host configuration and account information into the database.

UHA account management commands are intended for low volume transactions where immediate results are more important than the ability to manage a large number of accounts automatically. This difference in purpose results in a simpler architecture than that of other systems such as UAMS [10] and AGUS [7] that are capable of the high volume automated account processing required in a university environment.

High equipment turnover through vendor upgrade programs is typical in the EMSL in order to maintain a competitive research environment. The result is that many of our critical resources are running new (often beta) operating system releases. When cross-platform administration tools are provided by vendors as part of the operating system release, investment in commercial software that performs administration services on distributed systems will be practical in our environment. Considering products that can help administer the EMSL Distributed Computing Environment (DCE) cell now being deployed, maturing standards and the focus on cell management rather than host management bodes well for the use of comprehensive and supported commercial cell administration software.

Current Status

Components of the software described herein are in various stages of development. The OASIS schema, triggers, stored procedures, and server maintenance software have been in production use for about a year. The UHA load and show scripts have been deployed on SunOS and Solaris platforms for about six months; ports to IRIX and AIX are underway. The UHA account management scripts are currently being tested on SunOS and Solaris platforms; ports to IRIX and AIX will follow.

Future Work

Work is now underway to augment the spreadsheet front end to the database with a web-based interface. The UHA perl library modules for database access developed in the first release of the system make this task relatively easy.

The database contains the information necessary to create a useful set of electronic mail aliases for the EMSL mail hub. A similar service is provided by Moirra in the Athena environment. Again, the UHA infrastructure now in place will simplify this task.

The OASIS database is being extended to manage more of the information needed for EMSL operations, including staff demographics, operating procedures, training and safety programs, publications data, network cable schedules, and so on. As this integration effort advances, data duplication problems diminish and direct access to the database for host administration functions becomes an even greater advantage.

In general, future UHA work will focus on development of interactive tracking and administration tools that can be placed in the hands of users and clerical staff. Administration services that control the configuration of multiple hosts are the purview of the experienced system administrator. These will be handled using the EMSL AHMS software, possibly extended to draw configuration information from the OASIS database in a manner similar to other systems referenced in the Related Work and Products section.

Summary

This paper described software and a database developed at the EMSL to help manage its computing environment of heterogeneous UNIX workstations distributed across the PNNL campus.

The architecture of the UHA system is very simple. A relational database serves as the central information repository. Initial data entry is done through a spreadsheet interface or by importing flat files. Host and account data is kept current by perl scripts running periodically on the distributed hosts, automatically scheduled by a *cron*-based mechanism. Account management is assisted by scripts that retrieve account template information from the database and execute platform-specific commands to perform the desired operation.

Availability

The UHA system is not currently packaged for distribution. Contact the first author (address given below) if you wish to inquire about gaining access to the source code.

Acknowledgements

This work was funded by the Environmental Molecular Science Laboratory construction project at the Pacific Northwest National Laboratory (PNNL). PNNL is operated for the U.S. Department of Energy by Battelle under Contract DE-AC06-76RLO 1830.

Author Information

Greg Thomas holds an M.S. in computer science from the University of Washington and a dual B.A. in computer science and statistics from the University of California, Berkeley. While in school, he held internship positions with Bell Communications Research, GTE Laboratories, Lockheed Missiles and Space Company, and Sun Microsystems. He joined the Pacific Northwest National Laboratory as a research scientist in 1991, where he has developed software for projects ranging from a graphical interface for human genome databases to a portable multi-language dynamic memory allocation library for computational chemistry applications. He can be reached at Pacific Northwest National Laboratory, Mail Stop K7-22, P.O. Box 999, Richland, WA 99352; or electronically at gs_thomas@pnl.gov.

Jim Schroeder has been involved with computer systems integration projects at the Pacific Northwest National Laboratory for over twenty years as a technical contributor and project manager. He can be reached at jo_schroeder@pnl.gov.

Merrilee Orcutt holds an M.S. in information sciences from Hawaii Pacific University and a B.A. in computer science from the University of California, San Diego. She joined the Pacific Northwest National Laboratory as a research scientist in 1993, and is currently the software engineering task leader for the Extensible Computational Chemistry Environment project. She can be reached at me_orcutt@pnl.gov.

Desiree Johnson graduated in 1991 from the University of Minnesota with a B.S. in computer science. She spent three years at the Pacific Northwest National Laboratory in the Applied Physics group working with heterogeneous database systems. She currently works in the Information Resource Management group at Norwest Mortgage in Des Moines, Iowa, supporting VLDB systems. She can be reached at Desiree.Johnson@nmb.norwest.com.

Jeff Simmelink graduated in 1990 from Eastern Washington University with a B.A. in computer science and mathematics. He spent five years at the Pacific Northwest National Laboratory as a UNIX systems administrator. He is currently a UNIX systems analyst at Itron, Inc. in Spokane, Washington. He can be reached at jeff.simmelink@itron.com.

Jack Moore left the Pacific Northwest National Laboratory in 1995 to join Itron, Inc. in Spokane, Washington. Reach him at jack.moore@itron.com.

Bibliography

- [1] Paul Anderson. Towards a High-Level Machine Configuration System. In *Proceedings of the Eighth Systems Administration Conference*, pages 19-26. USENIX Association, September 1994.
- [2] George A. Champine. *MIT Project Athena: A Model for Distributed Campus Computing*. Digital Press, Bedford, MA, 1991.
- [3] Michael A. Cooper. *Sysinfo Reference Manual*. University of Southern California, Los Angeles, CA, version 3.1.3 edition, March 1996.
- [4] Magnus Harlander. Central System Administration in a Heterogeneous Unix Environment: GeNUAdmin. In *Proceedings of the Eighth Systems Administration Conference*, pages 1-8. USENIX Association, September 1994.
- [5] Microsoft Corporation. *Microsoft Query User's Guide*.
- [6] Michael Peppler. *Syberperl Reference Manual*. ITF Management SA, Geneva, Switzerland, version 2.0 edition, October 1995.

- [7] Paul Riddle, Paul Danckaert, and Matt Metaferia. AGUS: An Automatic Multi-Platform Account Generation System. In *Proceedings of the Ninth Systems Administration Conference*, pages 171-180. USENIX Association, September 1995.
- [8] M. Rosenstein, D. Geer, and P. Levine. The Athena Service Management System. In *Proceedings of the USENIX Winter Conference*. USENIX Association, February 1988.
- [9] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proceedings of the USENIX Winter Conference*. USENIX Association, February 1988.
- [10] Roland J. Stolfa. Simplifying System Administration Tasks: The UAMS Approach. In *Proceedings of the Seventh Systems Administration Conference*, pages 203-208. USENIX Association, November 1993.
- [11] Sybase, Incorporated, Emeryville, CA. *Open Client DB-Library/C Reference Manual*, 1994.
- [12] Sybase, Incorporated, Emeryville, CA. *SQL Server Reference Manual*, 1994.
- [13] Sybase, Incorporated, Emeryville, CA. *SQL Server Utility Programs for UNIX*, 1994.
- [14] Sybase, Incorporated, Emeryville, CA. *Transact-SQL User's Guide*, 1994.
- [15] Transarc Corporation, Pittsburgh, PA. *AFS User's Guide*.
- [16] Stephen R. van den Berg. *Procmil Reference Manual*, version 2.81 edition, February 1993.
- [17] Larry Wall. *Perl Reference Manual*, version 5.001m edition.
- [18] Larry Wall and Randal L. Schwartz. *Programming perl*. O'Reilly and Associates, Inc., Sebastopol, CA, 1991.

Visualizing Huge Tracefiles with Xscal

Alva L. Couch – Tufts University

ABSTRACT

Xscal is a tool for visualizing the behavior of large numbers of comparable entities such as computers, routers, processes, users, etc. The input to Xscal is a table of numbers and strings, which can represent a system snapshot at an instant of time or an event trace of system activity over time. Xscal depicts the global distribution of values in one or two columns of that data. The user can then annotate the visualization to denote categories of behavior, after which Xscal can depict detailed information on entities in each category. Originally developed to study massively parallel computation, Xscal is applicable to a wide variety of system analysis tasks, from visualizing the process table or accounting statistics for a large server to studying the performance of a large network.

Introduction

Very often there is a problem with a large system or network that requires scanning a large amount of data for the answer. This data may be console output, process statistics, accounting data, or network performance measurements. While statistics on the data may be useful in isolated cases, in practice one must actually read the detailed data in order to reach a conclusion. Although very little of this data needs to be read, it is usually mixed in with much other data of little relevance. Our problem is to locate that data and interpret it to reach a conclusion, both quickly and reliably. Usually, this involves locating unusual behavior or occurrences in a sea of normal behavior and occurrences. Visualization is a particularly powerful technique for locating such unusual behavior.

There are many existing approaches to visualizing log files, all of which attempt to preserve the character of the log file as a stream of events. The paired tools `tcpdump` and `tcpview` [8] allow the analyst to dynamically filter the stream of output from a network monitor to eliminate packet records of little interest. The user interactively decides what events to keep in the event trace. SeeLog, part of AT&T's software visualization suite SeeSoft, produces several depictions of a program log file. Events are colored in accordance with their type, so that the user can see at a glance where interesting events might be recorded. There are many other tools like `tcpview` and SeeLog, all of which view a logfile as a series of events and utilize color mapping, filtering, and zooming to navigate the event stream. But all of these tools are relatively specific to particular problem domains, such as networking.

Several very powerful and general-purpose visualization tools can be adapted to analyze networks if one is willing to interface system data to them. My parallel computation analyzer, Seeplex [2], and its relatives, Paragraph [4] and Pablo [7], can generate a variety of views of system performance for a network. However, one must first translate system data into a

form compatible with their data formats. In the case of Seeplex and Pablo, this is possible but quite involved. Paragraph is too domain-specific to parallel computers to generate much useful information. Survey data shows [6] that many intended clients for these tools consider their use too involved to be worth the effort of learning, even users working in the tools' original problem domain. Clearly, system administrators need something easier to use.

Xscal

Xscal is a visualization program that can display and analyze data for thousands of entities such as machines, networking hardware, or phone switches. Originally designed for analyzing massively parallel computations on networks of thousands of cooperating processors, Xscal is equally useful when we need to comprehend very large snapshots or trace files from a server or network. Xscal graphically depicts the global distribution of data, allows the user to group data by behavior, and exhibits original data by group. There is no limit to input file size except the physical memory of the host system, the speed of the processor, and the patience of the user. I have visualized 50000-line and larger files on a 128-megabyte Sun SparcStation-10 using this tool. Xscal is an X11 tool written using the freely available Openwin X11 toolkit, and is portable to a majority of X11 systems.

Xscal differs from its predecessors in several ways. Xscal emphasizes data views that remain usable regardless of the size of the data file. Scale-sensitive views that change in usability as the number of entries increase are de-emphasized. Xscal reads the simplest possible input format, unlike other tools with moderately to extremely complex requirements for input format. While the emphasis in other tools is simply upon depicting the data, Xscal allows the user to graphically annotate data to group or separate behaviors and can depict further views based upon that annotation. In modern parlance, Xscal is a 'highly interactive' data browsing interface.

Using Xscal

Input to Xscal is a table of columns of data delimited by spaces and tabs, where each column entry is either a number or an ASCII string. Depending upon how Xscal is invoked, the data file is interpreted either as a snapshot at an instant of time or as an event trace over time. In a snapshot file, each line of data describes everything known about a particular entity, where each column is assumed to contain commensurate measurements for the same kind of data. A trace file embodies many snapshots by describing each entity at several instants of time. Each trace file must have two columns, designated by the user, that denote which entity is being described and what time the data was recorded. All other columns are assumed to contain measurements or state information that become valid at the stated time and may be changed by the next entry (in time) for the same entity. Lines in the trace file are sorted into time order by Xscal and need not be sorted into any particular order beforehand.

Unfortunately, Xscal only knows how to read integers and floating point numbers, so that it cannot parse times expressed in date format. Entity descriptions, however, can be any text, including non-numeric strings. Below we will discuss ways to adapt output of common UNIX tools to Xscal's format.

Xscal's trace format is complex enough that a simple example may clarify it. The event trace file:

```
" time user processes
  1  couch 20
  7  sofa 10
 10  couch 10
 15  sofa  5
```

means the following:

- Before time 1 nothing is known about either couch or sofa.
- From time 1 to before 10, couch has 20 processes.
- From time 7 to before time 15, sofa has 10 processes.
- From time 10 to the end of time, couch has 10 processes.
- From time 15 to the end of time, sofa has 5 processes.

In other words, the input file above corresponds to the following sequence of actual values, where values in each line are valid until the next time:

time	couch procs	sofa procs
1	20	-
7	20	10
10	10	10
15	10	5

This format may seem complex, but it was derived from much experience designing and parsing much more complex forms [3]. I believe it to be the simplest

possible form to which any event data can be converted.

Xscal's Views

In either snapshot or trace mode, Xscal obeys a consistent strategy in displaying data. First it depicts a global representation of all measurements either at an instant or over all time and awaits user input. The user may then choose colors and textures to designate particular ranges of values. Xscal can then generate subsequent views depicting statistics and detailed data on selected ranges. In particular, if a range of values indicates undesirable behavior, one can immediately obtain full details on the conditions surrounding that behavior.

Viewing Snapshots

In snapshot mode, every line of the input file represents a summary of measurements for a distinct entity. For a chosen column of the data, Xscal produces a depiction of both values and frequencies of data in that column (Figure 1). The vertical axis represents value while horizontal axis represents individual entities. On the vertical axis, there is a hash mark for each distinct data value. On the horizontal axis, entities are sorted into order of increasing value from left to right, and there is a hash mark between each two groups of entities with identical data values, so that whitespace between hash marks indicates the frequency of a particular value.

These two axes are combined into an X-Y plot that represents the distribution of values for an entity. This plot is created by plotting a point for each entity with horizontal offset equal to the entity's position on the horizontal axis and with vertical offset corresponding to its value on the vertical axis, filling in gaps so that these points become a horizontal line whose vertical offset represents a value and whose horizontal extent represents the value's frequency. These horizontal lines (of entity measurements) are joined end-to-end by vertical lines whenever values change, so that each vertical segment represents the difference between two adjacent values in sorted order. The line thus drawn always proceeds from bottom left, simply increasing, to top right of the diagram.

The shape of this 'characteristic line' indicates the global distribution of values for all entities.

1. A diagonal proceeding from bottom left to upper right denotes that values for all entities are uniformly distributed (but perhaps randomly distributed) over a given range of possible values.
2. A 'step' which begins at bottom left, proceeds horizontally to the middle, then vertically to top middle, then along the top to the right, denotes that there are two distinct values that occur with roughly equal frequency, one small and one larger.

3. A horizontal line denotes that all values are the same.
4. A line that rises quickly at the left and then levels off denotes that most values are relatively large, because the frequency of larger values is greater than that for smaller ones.
5. A line that stays low and rises near the right denotes that most values are relatively small.

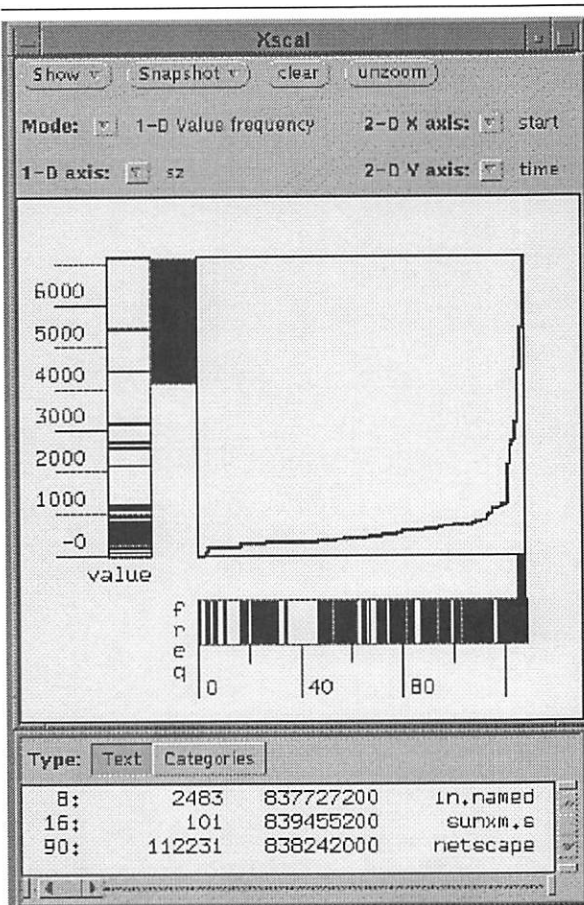


Figure 1: A snapshot of server memory usage based on ps command output.

The 'characteristic line' shows data distribution regardless of the structure of input values. If entities assume a small number of discrete states, it looks like a staircase where the width of each stair indicates the frequency of that state. If entities assume many distinct values, the line looks like a continuous curve.

Be not deceived, however: this depiction is very different from a normal X-Y plot. In particular, a characteristic line for a linear sequence of values looks almost identical to a characteristic line for white noise. By sorting measurements, the snapshot view effectively suppresses our knowledge of noisiness in the data, in order to coherently and understandably depict the global distribution of values. Any view that depicted that noise (such as a standard X-Y plot without sorting) would obscure the data distribution.

This depiction is a 'figure of merit' [5] for one system parameter, in the sense that one can tell, at a glance, whether the distribution of the parameter is normal or exceptional. To fully understand this, consider what happens when each entity is a process on a timesharing server and the depicted value is memory utilization.

1. A diagonal proceeding from bottom left to upper right means that a given process is equally likely to have any memory utilization from small to large.
2. A 'step' which begins at bottom left and vertically up the middle means that half the processes are memory hogs (with the same memory utilization(!)) while the other half are low-impact.
3. A horizontal line means that all processes have the same memory utilization (and something is very wrong with your server).
4. A line that rises quickly at the left and then levels off means that almost all processes consume a lot of memory.
5. A line that stays low and rises near the right means that a few processes are hogs and the rest relatively low-impact. This is also the characteristic appearance of a statistically normal distribution of values with a small standard deviation.

From this diagram, one can tell at a glance what action to take to improve system performance if memory is in short supply.

- In cases 1 and 4, there is very little that can be done, because a large number of processes are contributing to the problem. It is time to buy more memory or servers.
- In cases 2 and 3, something is probably seriously wrong with the server and it is time to reboot. Perhaps there is a runaway self-replicating process.
- In case 4, very few processes are causing the problem so it is perhaps time to ask a few users to be more courteous.

Of course, one could determine this from looking at the output of ps or top, but scanning this diagram shows much more than a printout of numbers.

Note that since the impact of multiple processes is the sum of their memory utilizations, the *area under any curve segment* has units of memory utilization \times frequency, which is cumulative memory utilization. This is proportional to the total impact upon the server for that segment.

Figure 1 depicts an actual Xscal depiction of memory utilization for all processes on a UNIX server. This graph is a trough curving upward from the base, implying that a few processes taking most of the memory (which is a fairly normal state of affairs). To determine what these processes are, we distinguish them by selecting a texture from a texture palette and painting

that texture onto the region of the Y axis for which we wish detailed data. The figure shows the resulting black rectangle. Then we ask for a textual view of the data, shown below the global view in the figure. This view shows only the offending processes we selected, sorted by memory consumption. The text view identifies several memory hogs.

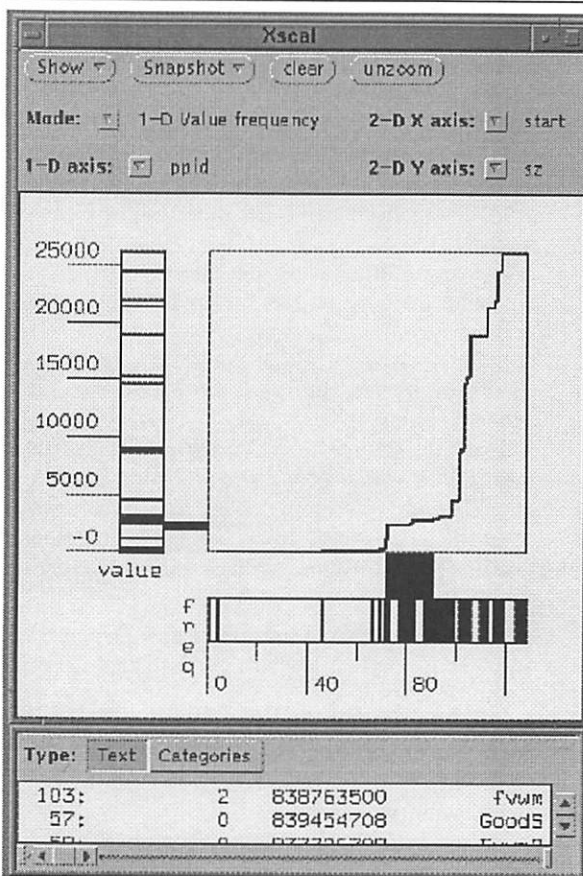


Figure 2: Displaying the distribution of parent process identifiers for all processes gives surprising insight into usage patterns for a server.

Figure 2 shows the distribution of parent process identifiers for all processes on a server. This distribution has an unusual shape consisting of multiple steps. Closer examination shows that each step represents an interactive session for one user, because all the user's processes are children of the same login process. Large steps are X11 sessions and small steps are tel-nets. Thus a datum that usually indicates little of interest when listed in a `ps` command, when shown in this way, tells one a great deal about how the server is being utilized.

The user can zoom in on details in the global view by selecting a rectangle of the view to magnify with the mouse. There is no limit to the detail of zooming, to the limit of floating point resolution. While zooming, scroll bubbles on the top and right hand sides of the view show one's relative position within the view.

Studying Correlations

Xscal can also be used to study the relationship or correlation between kinds of measurements. It does this using a standard two-dimensional scatterplot, where the user can choose the columns representing each axis. For example, Figure 3 represents a comparison of memory utilization and CPU time utilization for all processes, where memory utilization is depicted on the Y axis and CPU utilization on the X axis. Two-dimensional scatterplots can be explored just like one-dimensional distributions, except that one must select a rectangle of points for detailed study.

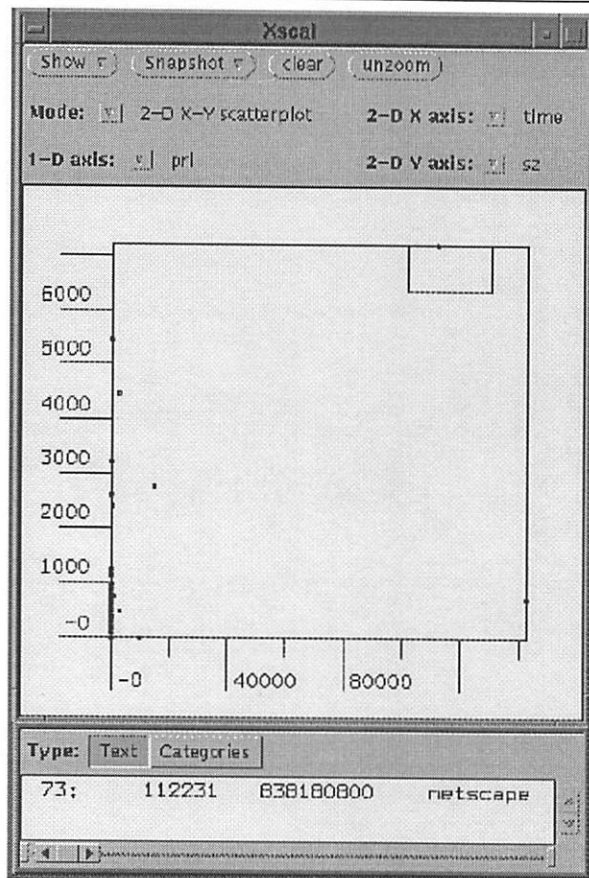


Figure 3: A two-dimensional scatterplot of memory utilization (Y) versus cumulative CPU utilization (X) for all processes on a timesharing server shows two runaway processes I would perhaps have missed otherwise.

When I casually generated this figure in writing this paper, I discovered that my server contained two runaway processes. The data point at the top right represents a process with extremely large memory utilization and total CPU utilization. I selected it by drawing a rectangle around it, so that detailed data was shown for it. Closer examination showed that it had been running for weeks. The data point at bottom right is also a runaway, but its impact is solely upon CPU usage, not memory.

Displaying Time Series

Xscal's second form enables display of time series data by interpreting input lines as events rather than states. Each line has a time when it occurred, an entity describing where it was measured, and a series of measurements for that time and place. There are no other restrictions; Xscal re-sorts input internally so it can be in any order, even backward. A measurement for an entity is valid from the time stated in its line to the time of the next measurement for the same entity.

Suppose we invoke Xscal on a login history, where entities are people (uids), time is wall clock time, and state is average memory consumption (from acctcom). Since the output of acctcom is not suitable for Xscal, I first convert it using a Perl script as described below. Of course, one can write the conversion script in any general-purpose programming language.

When invoked on a time series, Xscal first provides a scatter-plot of values versus time (Figure 4). The plot represents our total memory usage on a single server between 1 am and 11 pm on a single day. The vertical axis is again memory consumption, while the horizontal represents time. Entities are particular users. Each dot represents how much memory was being used, on average, by a particular user at a particular time.

To learn more about this plot, we click on a 'busy' time in the scatter-plot to reconstruct textual information and the value-frequency plot for the chosen time. These work just as if we were studying a single snapshot, but only show that subset of the event trace that determined measurements for the chosen time (events before and including that time). Using this view we mark two ranges of measurements for further analysis: we use the mouse to associate 'relatively high' memory utilization (in the upper third of the Y axis) with a dark texture, and 'moderate' memory usage (in the middle third) with a lighter one. These textures will then be used by Xscal to depict the behavior of each entity, both at an instant and over time.

Using this categorization, Xscal can show a frequency histogram summary of entity behavior over time (Figure 5, top right). In this histogram the horizontal axis is time and the vertical axis is frequency. The height of a textural element at a given time represents the number of entities having behavior corresponding to that texture, i.e., moderate or high memory utilization. While the global depiction of behavior seems to indicate a busy day, the thin dark line in the histogram shows that only at most three users are using a really large amount of memory at any time.

To exhibit behavior of particular users, Xscal can also depict a timeline for each entity (Figure 5, bottom

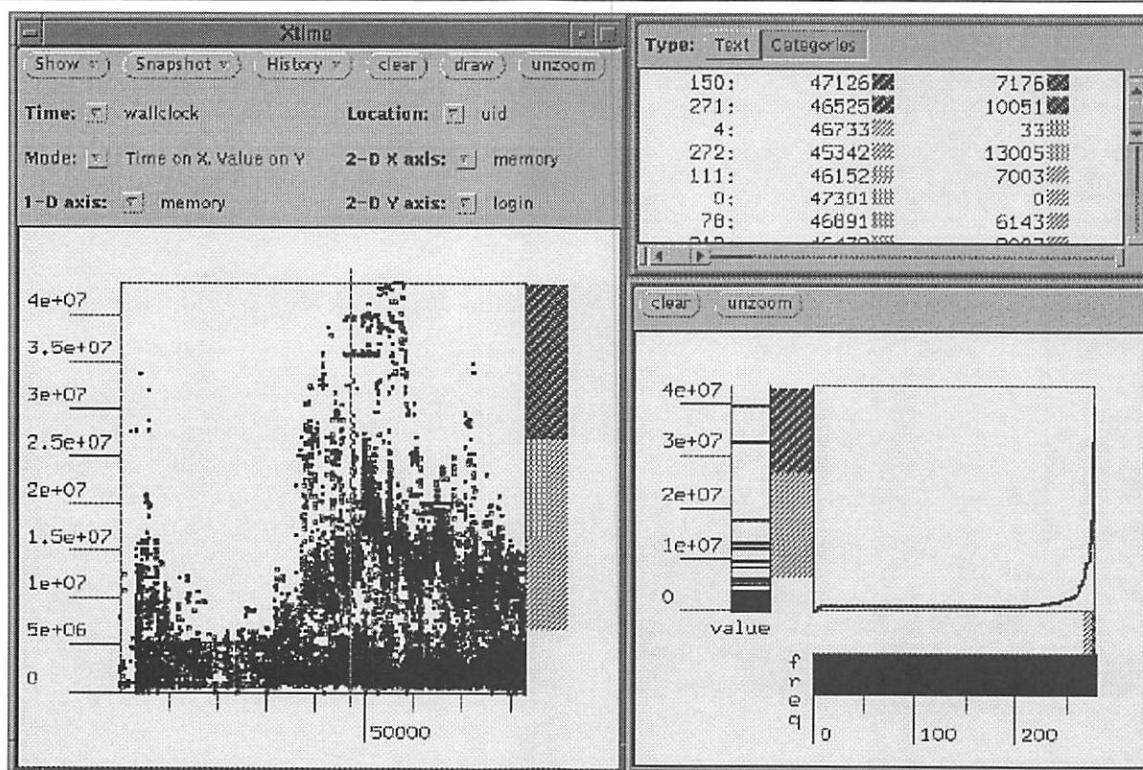


Figure 4: Average memory usage for each user on our server between 1 AM and 11 PM on a single day. At left is a plot of time on the X-axis against user memory usage on the Y-axis. Selecting a time reconstructs a snapshot of conditions at that time, shown at right in textual form and as a distribution.

right). This shows that although we might be concerned about the global graph, our users actually individually utilized a lot of memory for relatively short periods. It is perhaps time to buy a new server.

The strength of all of Xscal's views is that it is easy to distinguish relatively normal behavior from exceptional behavior and quickly detailed information on the exceptions. Using these views, we can very quickly answer the two relevant questions about a resource conflict problem: when and who!

Theory of Xscal

Xscal's views are based on principles discovered in my study of how to visualize the behavior of massively parallel programs [2]. One can create an infinitude of Xscal-like views by following several basic principles.

A *scalable view* is one whose form, size, meaning, and use do not vary with the amount of data it depicts. A view whose size changes is instead a *scrollable view*, because it eventually becomes larger than screen size and must be viewed in parts by scrolling. Any view that depicts distinct entities is by nature scrollable, so that scalable views must instead depict data distributions.

Using three basic building blocks, the time axis, the value axis, and the frequency axis (as depicted in

the snapshot global view), one can depict many distinct scalable depictions. The value and frequency axes are typically both necessary and somewhat dual to one another. For a datum that takes only a few distinct values, frequency usually indicates more than value, while for a datum that takes many values, value shows more than frequency. Both are present in the basic global parameter view in order to effectively represent all possible data distributions.

While Xscal more or less exhaustively covers the possibilities for two-dimensional scalable views, these principles give rise to many useful three-dimensional views that Xscal does not yet provide. The snapshot views of a datum, viewed over time, form a surface where one axis is time, another value, and a third is frequency. The evolution of the parameter can be seen at a glance in this view. Another example is a two-dimensional scatterplot of values depicted over time.

Converting File Formats

Unfortunately, Xscal was designed to analyze traces of parallel algorithms, which can be made more consistent in format than standard system log files. Fortunately, Xscal can take input from a UNIX pipe, so that Perl scripts can be used to translate between UNIX output and Xscal's desires. Since Xscal does not limit the user to a particular number of columns of

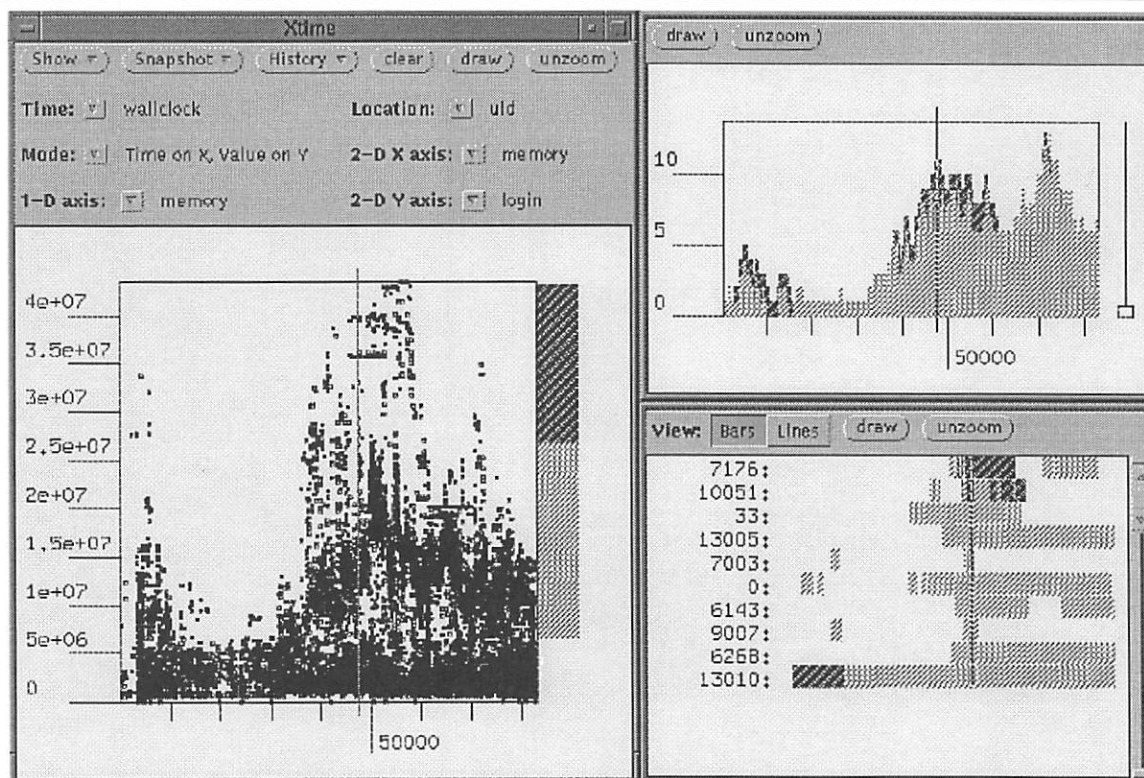


Figure 5: Given categorizations of memory usages as high (dark texture) and moderate (medium texture), Xscal can show category frequency over time (top right) or timelines for particular users (bottom right). We see, not at all surprisingly, that most memory utilization is due to the activities of three users.

input, a timely strategy is to tack new columns onto the beginning of each line that contain Xscal-readable input. In the future, I plan to make domain-specific modifications to Xscal itself that will allow it to parse such things as UNIX date strings for syslog visualization.

There are two translation problems, one of which is easy and the other of which is quite difficult. If the input file already has a consistent columnar form in which each column has a consistent meaning, translation is relatively straightforward. This is true for process data, accounting data, filtered network traces containing only one protocol, etc. If input is amorphous in form, with different meanings to the same column in different lines, more extreme measures are needed to translate it into depictable form. In many cases depiction is impossible due to the nature of the data.

Translating a columnar form into an Xscal input requires a relatively straightforward, if ugly, Perl script. To generate the ps examples above, I added fields at the beginning of each line representing the numeric values of strings that Xscal cannot interpret as numbers. Xscal reads dates and times as simple strings, so I translate all times into seconds since the Epoch (Jan 1 1970, 0:00am GMT), using the Perl5 library module `Time::Local`. It is also necessary to translate strings like '10%' into their decimal equivalents (0.10) because Xscal will interpret the former as a string. I also place placeholders in empty fields, and change spaces embedded in fields to underscores so that fields will not be split up by Xscal; 'Jul 20' becomes 'Jul_20'.

The acctcom example was much more difficult to generate. I had to write a script that computes the global history of memory utilization for each user and represents that history in Xscal form. This required translating process statistics provided by acctcom into user statistics. For each elapsed time reference, a user's average memory consumption is the sum of the average memory utilizations for all processes that user is executing at that time. This may sound easy to compute, but in practice naive methods take too much CPU time to execute and the only efficient way to compute this requires advanced data structures such as balanced binary search trees.

Examples of format translation scripts used in preparing this paper may be obtained at <http://www.cs.tufts.edu/pub/xscal/lisa-96>.

Xscal Limitations

Xscal executes relatively slowly on large data files because it must initially sort and otherwise index the data. Its internal memory needs are roughly five times the size of the data file itself, in order to arrange for it to execute as quickly as possible. The input file to the last example contains 20000 lines of data and takes 20 CPU-seconds to read and render on an UltraSparc containing 128 megabytes of main memory.

Once input is read, rendering becomes much faster; repainting the screen takes about five seconds after each user request.

It takes time to tell Xscal what to do. As indicated above, translating an output format into Xscal's input format can be very involved, or even impossible. Once the file is translated, one still must tell Xscal which columns to interpret as what. This may seem like a small price to pay, except when a pressing systems problem must be solved. Then it takes too long to set up.

Xscal does not treat string input as well as it could. A string is just another way to indicate system state, and Xscal should treat it as such. One way is to construct a value axis for all strings, in which offset of a string on the axis corresponds to the ordinal of the string in a sorted list of all strings. Xscal could easily do this, but does not at this point. Xscal's internal assumption that all strings are at most 10 characters long is also a serious problem. Xscal should treat each line of data as an integral unit in textual displays, and keep the format the same in those displays as in an editor.

Conclusions

In any search for abnormality in a complex system, the most difficult problem is to define normality. Normality is easier to observe, feel, and experience than to define. In turn, a tool that allows one to compare experiences, normal and abnormal, is usually more effective in analyzing problems than any attempt to define normality rigorously. Of course, one has to choose views that by their nature look different in normal and abnormal situations. This is the driving philosophy and motivation behind all visualization of complex systems.

Several times, Xscal has saved me time in looking at systems data. I initially tried it on accounting data to settle a dispute about server planning. One faction claimed that the University's mail server was slow because some students were abusing their privileges. Xscal showed immediately that not only were the students all using roughly the same amount of resources, but also that resource problems stemmed from use of the system for administrative tasks.

Xscal allows several kinds of comparative analysis to be done easily and quickly. However, there are absolute limits to the utility of any general purpose tool in this problem domain. To be utilized regularly, a visualization strategy has to be *really easy* to employ, and Xscal, while very powerful, is still too difficult to use regularly. Data conversion is relatively difficult, setup takes too much time, and there is no provision for heterogeneous data sets where measurements for differing entities are not necessarily comparable.

This last problem makes it quite difficult for one to use Xscal or its relatives to analyze a network. Sure, one can understand the function of 1000 X-terminals

(of the same kind(!)) but what if one has 20 Suns, 10 SGI's, and 20 Dec workstations? Data for these stations is not necessarily comparable. Effective techniques for visualizing and understanding the performance of scalable heterogeneous networks are as yet undiscovered.

Xscal's principles, however, can be easily applied to domain-specific tools for viewing particular kinds of system data. For example, I can envision a process table lister (like top) based on Xscal, where several value-frequency views show the distribution of CPU consumption, memory consumption, etc. Xscal's coloring and texturing operations are too powerful; all one needs is an ability to move a scroll bubble around on the entity axis and examine entities in the order specified by the values. I can envision a special-purpose accounting viewer with the ability to reconstruct the distribution at a particular time, scroll through it using the value-frequency view as a scrollbar, and construct frequency histograms according to some automatically defined value range scheme. In retrospect, Xscal's tolerance for user-defined ranges may well be a weakness, because of the time it takes a user to set up such a color mapping scheme.

This makes it quite clear that Xscal should change from being a tool to being a library of visualization routines one can apply to different problem domains. I plan to work on this problem, and would welcome input on problem domains to which Xscal techniques could be applied, as well as opinions about the appropriate software platform to use in developing that library.

Availability

Xscal is freely available from <ftp://ftp.cs.tufts.edu/pub/xscal>. The current revision is 2.0Beta, which supports floating point. Revision 1.2 is limited to integer input. Source code for filters utilized in preparing this talk are available from <ftp://ftp.cs.tufts.edu/pub/xscal/lisa-96>.

Author Information

Alva L. Couch was born in Winston-Salem, North Carolina where he attended the North Carolina School of the Arts as a high school major in bassoon and contrabassoon performance. He received an S.B. in Architecture from MIT in 1978, after which he worked for four years as a systems analyst and administrator at Harvard Medical School. Returning to school, he received an M.S. in Mathematics from Tufts in 1987, and a Ph.D. in Mathematics from Tufts in 1988. He became a member of the faculty of Tufts Department of Computer Science in the fall of 1988, and is currently an Associate Professor of Electrical Engineering and Computer Science at Tufts. In 1996 he received the Leibner Award for excellence in teaching and advising from Tufts. He has assisted in maintaining the Tufts computer systems for Computer

Science teaching and research since 1985, when he was a Ph.D. student, and is currently responsible for maintaining the largest independent departmental computer network at Tufts. He can be reached by surface mail at the Department of Electrical Engineering and Computer Science, 161 College Avenue, Tufts University, Medford, MA 02155. He can be reached via electronic mail as couch@cs.tufts.edu. His work phone is (617)627-3674.

References

- [1] Alva Couch, *Graphical Representations of Program Performance on Hypercube Message-Passing Multiprocessors*, Tufts University Department of Computer Science Technical Report 88-4, April 1988.
- [2] Alva Couch, "Categories and Context in Scalable Execution Visualization", *Journal of Parallel and Distributed Computing* 18, August 1993, pp. 195-204.
- [3] Alva Couch and David Krumme, "Portable execution traces for debugging and performance visualization". *Proc. Scalable High Perf. Computing Conference*, IEEE Computer Society Press, 1991.
- [4] M. Heath and J. Etheridge, *Visualizing Performance of Parallel Programs*, ORNL Tech. Rep. ORNL-11813, Oak Ridge National Lab., Oak Ridge, TN, 1991.
- [5] Michael F. Morris, "Kiviat Graphs - Conventions and 'Figures of Merit'", *Performance Evaluation Review*, Vol 3, No 3, pp. 2-8, Oct. 1974.
- [6] C. M. Pancake and C. Cook, "What Users Need in Parallel Tool Support: Survey Results and Analysis," *Proc. Scalable High Performance Computing Conf.*, 1994, pp. 40-47. <http://www.nero.net/pancake/surveys/94-80-3.html>.
- [7] Dan Reed, R. Olson, R. Aydt, T. Madhyastha, T. Birkett, D. Jensen, B. Nazief, and B. Totty, "Scalable Performance Environments for Parallel Systems", *Proc. Sixth Distrib. Memory Computing Conf.*, IEEE Computer Society Press, 1992.
- [8] Tcpview: the tcpdump network trace viewer, available in <ftp://ftp.ac.washington.edu/pub/tcpview>.

Using Visualization in System and Network Administration

Doug Hughes – Auburn University

ABSTRACT

Unix systems have numerous tools that generate copious amounts of data about performance, security, process status, networking and sundry other things; this usually results in the output of raw numbers or text. Visualization of this data can lead to useful insights. Examples include: graphing performance data, customizing tools to make a complex operation easier to understand, and correlating log events according to user-defined rules or patterns. Using visualization in daily activity can help the brain recognize normal and aberrant behavior, make complex tasks easier, and improve workplace efficiency.

The goal of this paper is to detail how simple rapid-prototyping, GUI tools can be used to quickly develop applications to visualize data. This paper also describes the application of visualization to areas of system and network administration. I will attempt to illustrate my own results and experiences developing visualization tools in Tcl/Tk [1] (with various extensions). The design and implementation of four such tools will be discussed and used as examples of visualization applied toward system and network administration. Some have surpassed their original design criteria and provided unanticipated side-effects that enhance their utility.

Introduction

Though the precise definition of visualization is often debated, it is customarily used as a way to express information in two or three dimensions. The advantage of visualization is that it takes raw data and manipulates it such that recognizable patterns begin to emerge, or presents it such that a certain task becomes easier to understand or more efficient. Visualization can also be used for prediction and intuitive troubleshooting, once sufficient data has been collected. Through the use of rapid prototyping GUI tools like Tcl/Tk, Perl [2]/Tk, Python, etc. one can develop simple visualization tools that can help in various system and network administration tasks.

By nature, visualization works best with a GUI interface. According to Jakob Nielsen [3], there are five aspects to usability of a GUI interface: easy to learn, efficient, easy to remember, relatively error free or error forgiving, and pleasant to use. Because of the user-definable nature of the tools that I will be describing, and the target audience (myself, my co-workers, and others in my technical field), I will be concentrating on the first three aspects. This may cause the tools at times to be plain in appearance and unforgiving when used in unexpected ways, but future iterations and extensions will work to correct any deficiencies. All these tools have been designed in a language which makes individual tailoring quite easy.

Currently Auburn University College of Engineering uses several tools written in Tcl/Tk to indicate health and performance of networks and machines, to make certain tasks more manageable, and to troubleshoot when problems occur. The tools I will be

focusing on include a security analysis tool, a server CPU pie-chart tool, a SPARCstorage Array disk visualization, placement, and optimization tool, and briefly, a hub analysis tool.

Motivation

These tools have all been designed to make our jobs easier and more productive. Without these tools, perusing the raw data streams would be a time consuming and frustrating process. They help us to always be aware of events as they happen. Being a part of a support organization, it's important to at least give the illusion to your users that you know about a problem before they report it. If only the phone would answer itself when a server went down...

tklogger

We use TCP Wrappers [4], klaxon, tocsin [5], and other customized daemons and programs to log security events to a centralized, secure, limited-access machine via syslog. Originally, all of our syslog data was simply posted on the console of a machine as it arrived, without any organization or correlation possible. This resulted in one of two undesirable situations. First, if the console window was on top, a portion of the screen space was being used; if nothing was happening, that screen space was wasted. Second, if the console was in the back, the information was largely lost behind another window.

Tklogger was developed as a real-time security analysis tool. The goal was to have a way to correlate high priority and low priority traffic based on syslog data files and regular expression matches. It was determined that events should also be displayed in multiple colors to facilitate their interpretation at a glance and

without having to scrutinize the text (color event classification). Subsequently, a quick glance at the window could determine the security status of the hosts and the network. It was also decided that when important events occurred, the window would automatically come to the front. This way the administrator using the machine (myself) would not always have to keep a third eye on the console, nor would useful screen space be wasted. Other features such as search capability, scrollbar, and easy configuration were added as needed. It was originally constructed a few hours a day over the course of a week. Little modifications were made for the first few months, but the general functionality remained relatively unchanged.

cpupie

Cpupie was inspired by an object oriented Tcl/Tk pie-chart extension called tkpiechart [6]. Out of this simple demo came the idea for representing the CPU states of our servers as pies. The already widespread practice of breaking up a CPU into four components (idle, wait, system, and user) made its use in this capacity immediately apparent. Thus, displaying the CPU trends of our servers was imagined (and since proven) to be a useful symptom of server health. Mr. Fontaine (the author of the pie-chart widget) and I collaborated on the features of tkpiechart to get it to its present state. Excluding that work, the initial design phase for cpupie was approximately 16 hours.

ssa

Ssa was born in an afternoon of frustration. We have two SPARCstorage Arrays [7] attached to two servers. At the time, each had 12 disks that were almost completely filled with approximately 14 file systems on RAID-5 volumes. Also at the time, three new disks had come in for each array to add more file system capability to our home directory servers.

Since the array model that we own is already divided into six SCSI controllers of up to five devices per controller (Figure 1), it is convenient to purchase disks in groups of six to maximize the advantages of a six disk RAID-5 stripe. After we exhausted the capacity of our existing disks we were left in a bit of a conundrum: how do we take three disks, add them to the array, and arrange them such that no two stripes share a disk, busy stripes do not share a controller, and all disks are evenly used without overburdening the busy stripes? (The astute reader may note that with three new disks it is probably impossible to not have at least one volume that has two stripes on the same controller using a six disk stripe given that all disks will be evenly utilized.)

Without this tool, we would have had to write all configurations on a white-board in text with volume sizes and layout then manually arrange things by erasing, moving, and updating all totals without being able to see the final results until implementation. This approach was fraught with peril, was essentially brute-force and may have taken us hours for each array. The

ssa tool was written to speed up this process for this and all future instances of new disk arrival, removal, or failure. The first drag-and-drop, user-hostile version was made available in four hours. Since the creation, our decreased time spent calculating movements has more than made up for this design time.

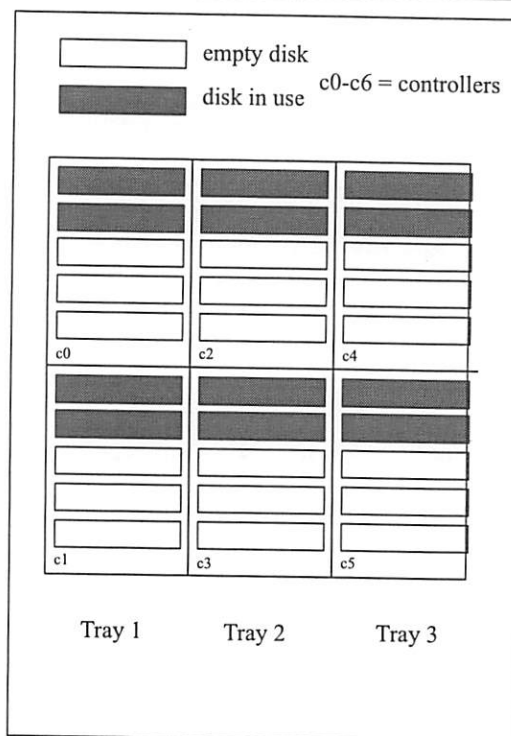


Figure 1: Array physical layout

hphubwatch

Finally, hphubwatch was written at a time when slowness problems were being experienced on several of our 30 networks. We required tools to monitor and analyze our HP hubs, but our budget was extremely tight. This tool uses SNMP [8] to gather information from HP AdvanceStack hubs and display it graphically in real time. It was written in a few hours and is fairly small and simple.

Program Implementation and Usage

tklogger

As already mentioned, the output of tklogger is contained in two windows. The latter contains high priority events and the former contains low priority events. An example window layout is in Figure 2. Normally, each text line would be displayed in the color associated with that event. An event is represented in user-defined colors to enable easily recognizing the type of event when it occurs without actually reading the corresponding text. In this way, a user can tell at a glance when something bad has happened or when an unusual pattern occurs. Further investigation may be warranted, particularly if an alarm color is displayed (anything matching red is high priority).

The idea for using colors to represent events occurred to me one day while experimenting with a Tcl/Tk text widget and seeing how easily it could be configured to display lines of text in multiple colors and/or fonts. The idea for using a Tcl/Tk text widget to display time sensitive logging information in colors blossomed out of that.

Augmenting the visualization is a search capability that allows one to highlight matches in place or display them in a separate window. Menus are available for users to adjust scrollbar capability, save events to a file, reload the configuration file, adjust the search options, and perform other actions. One can also pause the polling to examine an event more closely before resuming.

When an alarm event occurs the window immediately de-iconifies itself if necessary and rises above all other windows on the screen. A recent addition has allowed it to also perform other actions such as sending mail, paging, ringing the keyboard bell, or executing a user defined function written as a Tcl procedure. We have tklogger running at all times monitoring and displaying various events.

With proper configuration, tklogger can be used to monitor many log files concurrently. Each file may be given a base priority for all records. These base priorities can then be overridden with regular expressions. A Sample configuration file is shown in Figure 3. **File** directives specify the log file to poll. **Color** directives give a priority (high or low – based on color choice) that is the base priority for events in that file. All events appended to the end of a file will have this base priority color. Files that do not have a **color** directive will be examined for **match** expressions.

Regular or fixed expression **match** directives are used to override any applicable base priority, or to execute a command. **Ignore** directives are used to elide base priority information that may not be necessary (e.g., debugging information from sendmail).

```
file auth /var/log/authlog
file daemon /var/log/daemon
file local0 /var/log/local0.info
file local1 /var/log/local1.note
file local2 /var/log/local2.warn
file local3 /var/log/local3.note
file maillog /var/log/maillog
color local0 forestgreen
color local1 lightseagreen
color local2 magenta
color local3 red1
color auth red2
ignore NOQUEUE
match dlam red1
match mooneje {email page-doug
    {cshrc accessed}}
match help {playsound
    /home/ens/doug/sounds/chord.au
    orange}
match (LOGIN FAILURE) mediumvioletred
match (pgcntd|refused) red4
match portwatcher red3
match (vrfy|expn) violetred
```

Figure 3: tklogger rules

I would be remiss if I did not compare the usage of tklogger with two other popular log analysis tools. Contool [9] provides a way to perform actions when certain messages appear on the console, but lacks in areas such as event grouping and searching capability. Swatch [10] is another extremely useful and extensible

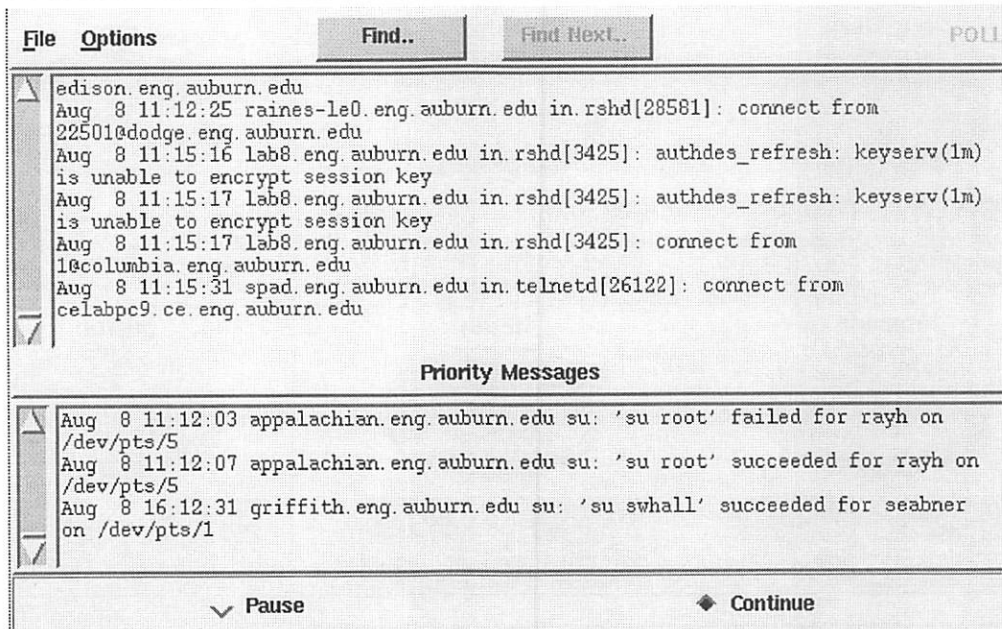


Figure 2: tklogger

log analysis tool written in Perl, which gives it all the power of that language. However, *swatch* is meant to be run in the background and I was looking for something more visual. Both tools are meant to process one input source at a time.

cpupie

To analyze our server CPU states, *cpupie* (see Figure 4) uses the *rstat(3)* portion of the *scotty* [11] Tcl/Tk extension. Since all Unix platforms of which I am aware support *rstat(3)*, the operating system independent nature of this tool was immediately attractive. The *scotty* extension also gave us the added advantage of client/server sockets so that only one machine was required to do the polling. Any other machines wanting to view CPU status of any subset of the servers could connect to the master machine for updates via a TCP/IP socket.

The *cpupie* program was designed with several simplistic features that we have found useful. The most commonly used features are available via buttons along the top of the window. It takes advantage of the native PostScript generation capabilities of the Tk canvas widget to output the current CPU states to color or black and white PostScript printers at the touch of a key. It has the capability to average the CPU states of all the servers over a user defined time interval. Finally, the polling interval and listening socket (for client/server operation) are user configurable. *Cpupie* is constantly being run by several people on two-headed workstations to monitor our servers.

ssa

Like *cpupie*, the *ssa* tool (Figure 5) is also implemented with a canvas widget for easily printing the current array layout. The most basic unit of operation on the canvas is a color filled rectangle. The disks are large rectangles, and each stripe (subdisk) of a disk is a smaller color filled rectangle. The stripes of a particular volume are all the same color across all disks. Mirrors of a volume are also the same color, but are stippled with a bitmap to indicate that they are mirrors and not the primary disk(s). Log disks have the same stipple as the mirrors do, but are usually invisible because of their tiny size, so a feature was added to artificially increase their size to identify disk location.

In order to understand the use of the *ssa* tool, one must also understand some of the nomenclature specific to the Veritas software that drives the SPARC-storage Array. Each physical disk is mapped to a VM disk which is divided into up to 16 subdisks via a virtual table of contents (VTOC). Subdisks are then amalgamated into a unit called a plex. This can be a RAID-3, RAID-5, or striped plex (in fact the basic functionality of a plex is analogous to a stripe in the conventional sense). These plexes can then be mirrored, concatenated, or combined with a log into a volume. In our case a file system goes onto the volume, though some places use them for raw databases partitions. The relationship between disks, subdisks and plexes is illustrated in Figure 6.

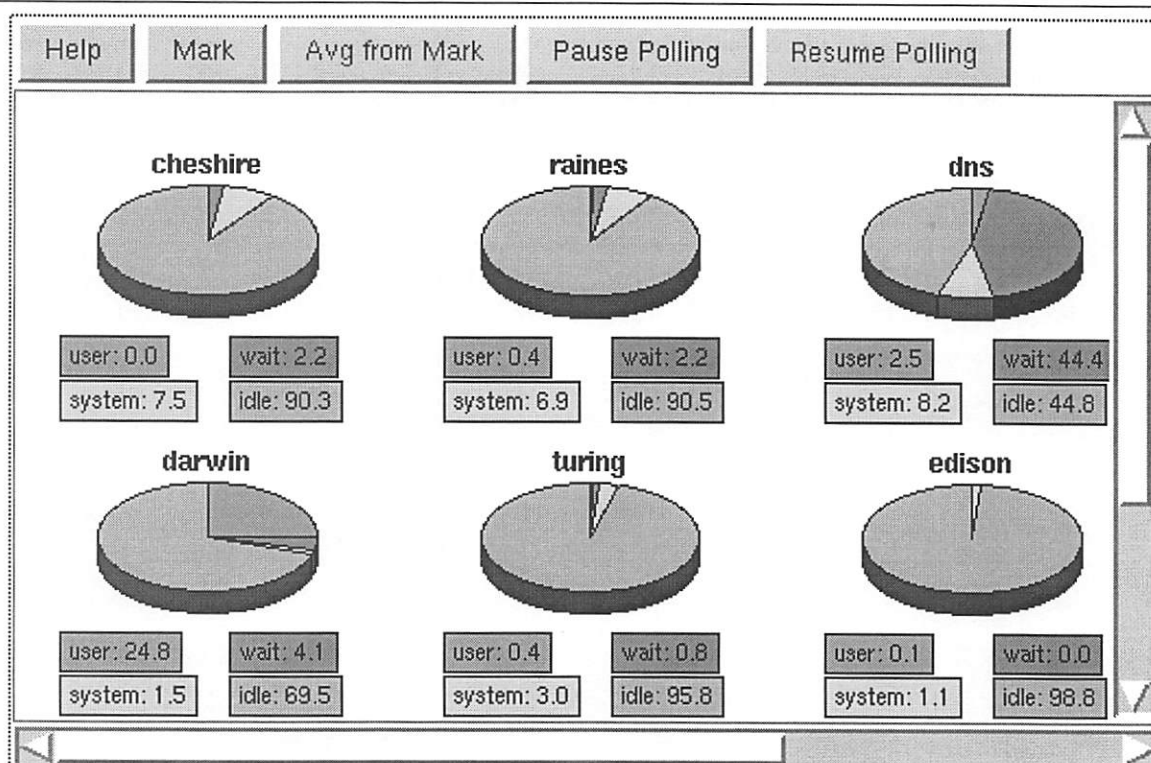


Figure 4: *cpupie*

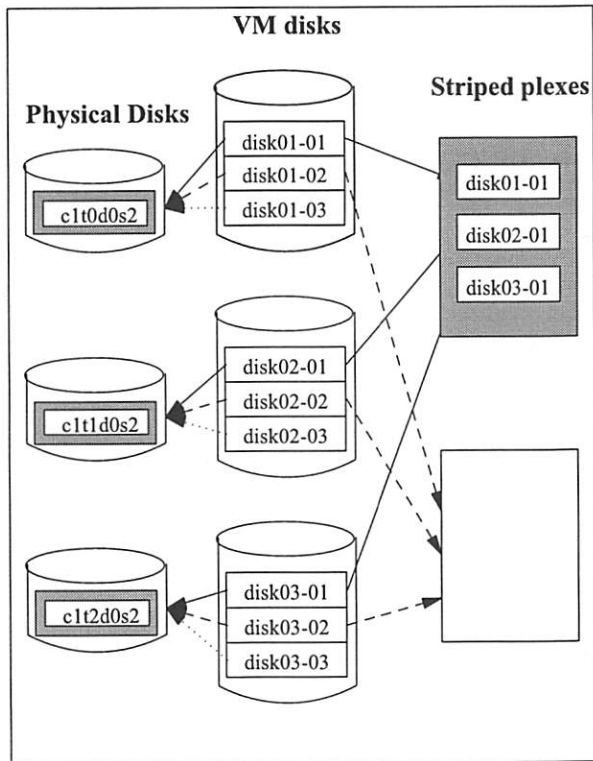


Figure 6: Volume Manager terminology

For people who don't use the Veritas Volume Manager (VxVM) GUI interface, a button also displays the sequence of commands necessary to configure the array when it has been arranged as desired.

This tool will not be valuable to people who do not have the Veritas software installed, but it is capable of running without an array.

The use of this tool is a bit more complex than the previously mentioned ones. Subdisks can be dragged from one disk onto another. New empty disks can be created with the click of a button to simulate the addition of a disk to the array. Also, there is an undo function, a way to determine the size of a sub-disk by clicking on it, and a way to determine the used and free space on a physical disk by clicking on it with mouse button 2. This tool is very VxVM specific, but it has served us quite well by allowing us to fine-tune placement of file systems on our SPARCstorage array and experiment with different configurations before implementation. It serves as a useful example of using visualization to simplify complex tasks.

hphubwatch

The hub watching tool is also implemented using scotty and, like tklogger, makes use of the color capabilities of the text widget. A portion of the window is shown in Figure 7. It currently assumes one machine is connected per port. The three columns following the MAC address correspond to administrative status, operational status, and media status respectively. The rest of the columns are explained in the legend of the figure. Information that has changed is highlighted in yellow. In the figure, the number of frames has changed on ports 5, 7, 8, and 9 (yellow) and the percentage of collisions to frames on port 8 and 9 is greater than 0 (red). Information that is deemed

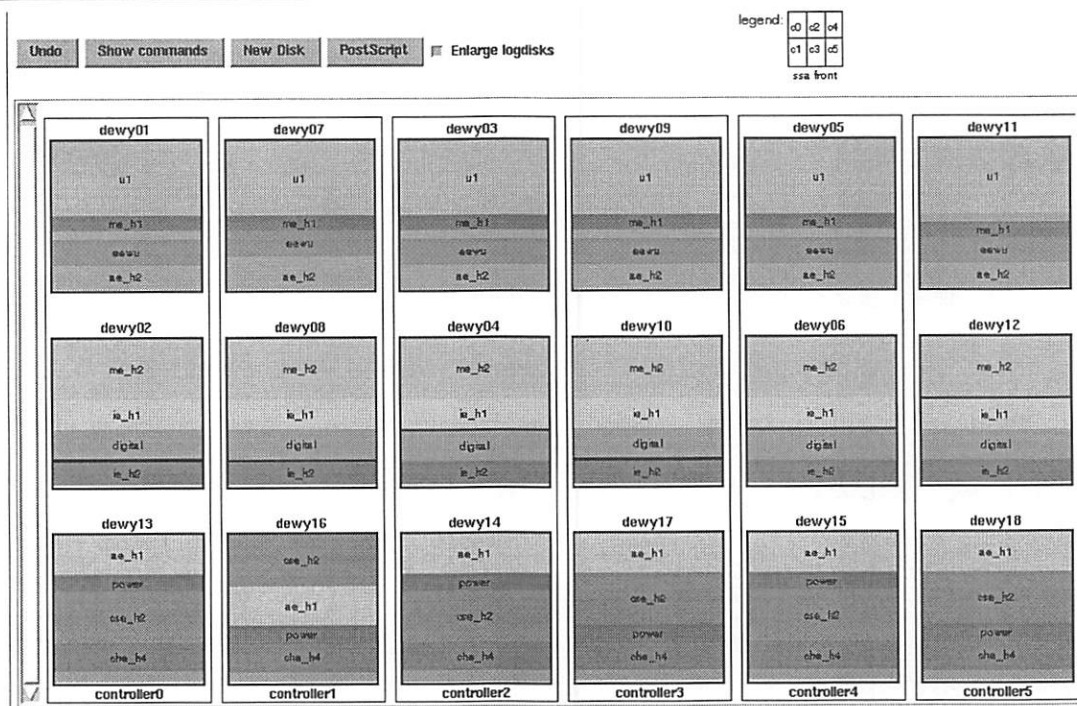


Figure 5: ssa

significant is highlighted in red. This information includes giants, jabbers, alignment errors, and other events of this nature which might indicate problems. The polling interval is user specified and the highlighted regions change after each polling interval. It is a very simple tool with narrowly defined operating parameters. We use it to troubleshoot networks through their hubs when problems occur.

Visualization Results

All of the tools have lived up to my expectations of their design. There have been serendipitous surprises as well. The paragraphs below will attempt to outline the expected results as well as describing how some of our expectations have been surpassed.

tklogger

Tklogger has already proved its usefulness at detecting many different types of events in progress including mail spam, port scanning, user account cracking, and misconfigured daemons. When a high priority event occurs, it has been beneficial to search back to find low priority events that may provide correlative information. We even have people using it for different purposes. I use it to monitor security while another person uses it to monitor WWW logs.

The multiple input file functionality has proven to be the most useful (non-visualization related) feature by allowing syslog generated events to be organized in multiple files by facility and priority. Tools that generate syslog information have the information forwarded to the secure log-host where it is stored in files according to simple syslog configuration rules. This facilitates searching and archival of logs based on administrator defined event groupings. This has been considerably easier than dealing with one large file.

cpupie

Cpupie began as a fun project to do with Tcl/Tk that could tell us generally how busy our servers were. It has since proven to be a useful forensic tool for diagnosing potential server problems before they happen. The CPU states convey information about other pieces of the system indirectly such as memory and disk usage.

The color green on any CPU is intuitively obvious. Our brains have been trained from childhood that green indicates that all is well. In this case green represents idle time. A green CPU tells us the machine has room to grow.

Yellow corresponds to system activity. Typically, yellow on our CPU's is 5-25% of the pie. When yellow gets higher than this, it is usually an indication to us that there is either a lot of process activity occurring (a daemon possibly gone amok) or a lot of memory in use for some reason. Our license server typically has 15% of its single CPU in yellow.

When a CPU exceeds 15% in the red state, we investigate. This usually indicates that there is a large amount of disk activity occurring. On our mail server this can be normal if a large mail list is being processed. On any other servers this usually means the machine is involved in some swapping or paging. Subsequent investigation leads to a culprit or the resignation that more memory is needed.

There is an interesting distinction between the user CPU state (displayed in blue) and the other three states. While resource usage typically balances across all CPU's, CPU time spent in user is easier to visualize on one processor of a machine. Therefore, on a machine with four CPU's, a CPU with 50% blue should be interpreted as 100% of two CPU's in use. On our quad-CPU machines the affect of this

A = Media Available		N = Not Available/Not Operational	
O = Operational		E = Administratively Enabled	
Med = Media Errors		Al = Alignment Errors	
Gi = Giant Packets		Rn = Alignment Errors	
Pt = Partitional Port		JP = Past Jabbers	
P = Port Polarity		I = Intruder	
FPS = Frames per second		%C = Percentage of collisions/frames	

	Mac Address			Frames	Octets	Med	Al	Tot	G	R	Collis	Pt	JP	P	I	FPS	%C
1	00:00:1D:04:A3:C4	E	O	A	96717	12320617	0	0	0	0	0	23481	0	0	0	0	0
2	00:00:00:00:00:00	E	N	N	0	0	0	0	0	0	0	0	0	0	0	0	0
3	00:00:00:00:00:00	E	N	N	0	0	0	0	0	0	0	0	0	0	0	0	0
4	00:00:00:00:00:00	E	N	N	0	0	0	0	0	0	0	0	0	0	0	0	0
5	00:00:C0:05:88:15	E	O	A	3689803	682703787	2	0	2	2	0	447600	0	0	0	0	0
6	00:00:1D:02:16:A3	E	O	A	57633	5160102	0	0	0	0	0	25559	0	0	0	0	0
7	00:00:F4:C8:BF:B1	E	O	A	199516	18354766	0	0	0	0	0	71680	0	0	0	0	0
8	00:00:F4:D0:32:62	E	O	A	536727	96050741	149	0	149	0	19	27493	0	0	0	0	0
9	00:00:F4:CA:6F:9D	E	O	A	55620	4788085	0	0	0	0	0	22812	0	0	0	0	0
10	00:00:00:00:00:00	E	N	N	0	0	0	0	0	0	0	0	0	0	0	0	0
11	00:00:00:00:00:00	E	N	N	0	0	0	0	0	0	0	0	0	0	0	0	0
12	00:00:00:00:00:00	E	N	N	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 7: : hphubwatch

demarcation is readily apparent. The blue slice of the pie is almost always at one of five levels: 0, 25, 50, 75, or 100. We can tell at a glance which compute servers would be candidates for new jobs. As our users write code to take advantage of multi-threading in modern operating systems, these distinctions may be less visible. In Figure 4, the machine **darwin** is a four CPU compute server with one CPU servicing a CPU-bound process and **dns** is a mail server (processing a mail list as it would happen).

ssa

Since our first addition of disks, we have learned that it is much easier to buy disks in groups of six to add to the arrays. However, we still use the tool when new disks arrive. In order to not overburden any controller or disk, we try to distribute things as evenly as possible. This means that when new disks come in we need to move subdisks and adjust free space to provide balanced access across controllers and disks. It has gone through several small enhancements since its creation in November of 1995 and is now under general release to help other users of Veritas Software around the world.

The SPARCstorage array tool makes our jobs a lot easier by allowing us to see at a glance where space is available. VxVM is very flexible at letting you customize your views of the array, but is not designed to allow one to experiment with new disk/volume layouts. It is a production tool. Once you commit to an action, you must wait until it is completed. With the storage array tool we have been able to reduce our configuration time from hours to minutes, save a lot of white-board space, print out (graphically) a picture of the current array layout, and see the actual commands that will be executed by VxVM when we are done.

hphubwatch

We have already identified several high traffic machines in work-groups on particular subnets and endeavored to further segment these networks, or install switches when required. In addition the tool has discovered, via high numbers of media errors, several old, flat, gray-satin cables that people had used to plug their computers into the network. It has also discovered the occasional improperly punched-down house wiring that has reverse polarity. In the hubs where security is enabled, it lets us track intruders on ports where undergrads may be trying to plug in their laptops or otherwise access the network in an unauthorized manner.

Conclusions

In my experience the genesis of new visualization tools has been 50% need and 50% inspiration. One of our most useful performance tools (cpupie) was inspired by a simple demo. Another (tklogger) was created as an experiment in log analysis. A third was created to meet a specific goal (ssa). The

remaining tool presented in this paper (hphubwatch) was created as an experiment and out of the desire to manage our hubs. All four tools have been in use for 6-18 months.

I have found that using different colors is a convenient way to express desired characteristics. The brain can distinguish between thousands if not millions of colors. More is not necessarily better, though. Experiment with your color choices and use those that work best on your application in your environment. I have experimented with fonts in different pitches and styles (e.g., italic vs. bold), but they do not have the versatility of color. Combining font changes with color changes has worked well. When you do use colors, try to use consistent strategies among tools. (i.e green is good, yellow may need investigating, red should be checked on immediately). Lastly, try to arrange contrasting colors between similar colors (stick a purple between your light seagreen and your medium seagreen).

Visualization tools help the system and network administrator to make sense out of complex data. Most any program that generates statistical or performance data can be visualized. Using visualization can provide a revelation out of an otherwise rudimentary set of unprocessed data. Patterns may begin to emerge which make management of resources, and trend analysis easier.

Visualization also can make complex tasks (e.g., file system rearrangement on a group of disks) easier by displaying the problem in an easier to understand or more intuitive format. Tcl/Tk has been particularly suited to these tasks by providing an easy to learn interpreted language with excellent publicly available contributions, and extensibility via C and C++.

Acknowledgments

I would like to thank my boss, Steve Henderson, for giving me the freedom to construct the tools about which this paper is written. I would also like to thank all the contributors of feedback and enhancements for their input. Finally, I would like to extend my deepest appreciation to all of the writers of the extensions to Tcl/Tk that made these tools possible, particularly Jean-Luc Fontaine, Karl Lehenbauer, and Jürgen Schöenwälder for contributing tkpiechart, TclX, and scotty respectively.

Availability

All tools will run on any Unix platform where Tcl/Tk and the relevant extensions can be compiled (which is most of them). Tklogger is self sufficient and should be able to run on Windows and Macintosh platforms as well (with minor modifications regarding file naming conventions). All tools are available for anonymous FTP at ftp.eng.auburn.edu in the pub/doug directory. A sample configuration file for tklogger (.tkloggerrc) is available on this server as well.

Tklogger is also available at the Coast security archives (<http://www.cs.purdue.edu/coast>). Documentation for each tool is included inside the file near the top.

Author Information

Doug Hughes got started administering Sun systems while attending Penn State University. He graduated in 1991 with a BE in Computer Engineering. From there he spent three years in GE Aerospace before and after the merger with Martin Marietta (pre-Lockheed, post-RCA) doing everything from software development to networking, systems administration, and internal consulting for client-server projects. From there he went to Auburn University College of Engineering where he now resides as the Senior Network Engineer. His interests include writing programs in scripting language to ostensibly make his job easier. He can be reached via U.S. Mail at 103 L building, Auburn, AL 36849, or via electronic mail at Doug.Hughes@eng.auburn.edu.

References

- [1] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison Wesley, Reading, Mass, April 1994.
- [2] Larry Wall, Randal Schwartz, *Programming Perl*, O'Reilly and Associates, Sebastopol, CA, 1991.
- [3] Jakob Nielsen, "Iterative User-Interface Design", IEEE Computer Society, *Computer*, Vol 27, n7, July 1994, pp. 32-41.
- [4] Wietse Venema, "TCP Wrapper: Network Monitoring, Access Control and Booby Traps", *Proc. 1992 USENIX UNIX Security Symposium*, pp 85-92.
- [5] Doug Hughes, *klaxon and tocsin - detecting port scanning*, 1995-96, unpublished tools, available via FTP at <ftp://ftp.eng.auburn.edu/pub/doug/>.
- [6] Jean-Luc Fontaine, *a Tcl/Tk pie utility*, 1996, available via ftp at ftp://ftp.neosoft.com/pub/tcl/alcatel/code/tkpiechart-*.tar.gz.
- [7] Sun Microsystems Inc., *Understanding Disk Arrays*, white paper, 1994.
- [8] Marshall T. Rose, *The Simple Book*, 2nd edition, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1994.
- [9] Chuck Musciano, *contool*, 1994, available via ftp at [ftp.x.org/R5contrib/](ftp://ftp.x.org/R5contrib/).
- [10] Stephen E. Hansen, E. Todd Atkins, "Automated System Monitoring and Notification With Swatch", *Proc. Nov. 1993 USENIX LISA*, pp 145-155.
- [11] Jüergen Schöenwälder, "scotty - a Tcl interpreter with TCP/IP extensions", *Proc. 3rd USENIX Tcl/Tk Workshop*, Jul. 1995.

How to Avoid Learning Expect -or- Automating Automating Interactive Programs

Don Libes – National Institute of Standards and Technology

ABSTRACT

Expect is a tool for automating interactive programs. Expect is controlled by writing Tcl scripts, traditionally a manual process. This paper describes Autoexpect – a tool that generates Expect scripts automatically by watching actual interactions and then writing the appropriate script. Using Autoexpect, it is possible to create Expect scripts without writing any actual Expect statements and without any knowledge of Expect.

Keywords: Autoexpect; Expect; interaction automation; Tcl

Introduction

Autoexpect is a tool that generates Expect scripts automatically by watching actual interactions and then writing the appropriate script. Using Autoexpect, it is possible to create Expect scripts without writing any actual Expect statements and without any knowledge of Expect. While this may sound useful only to beginners, even Expect experts now turn to Autoexpect because it is so effective at what it does.

Background

Expect is a tool for automating interactive programs. It is possible to make very sophisticated Expect scripts. For example, different patterns can be expected simultaneously either from one or many processes, with different actions in each case. Traditional control structures such as if/then/else, procedures, and recursion are available. A thorough description of Expect is found in [Libes95].

Expect's language facilities are provided by Tcl, a very traditional scripting language. (A thorough description of Tcl is found in [Ouster].) Traditionally, users write Expect scripts by studying the interaction to be automated and writing the corresponding Expect commands to perform the interaction.

Autoexpect is a program which watches a user interacting with another program and creates an Expect script that reproduces the interactions. In its simplest use, Autoexpect is quite straightforward. For example, consider an ftp session to ftp.uu.net. Normally, it would start out this way:

```
% ftp ftp.uu.net
```

At this point, the user would then interact with ftp. To have Autoexpect automate this, the only difference would be to start the interaction with this line:

```
% autoexpect ftp ftp.uu.net
```

The remainder of the interaction would be the same – the user would interact with ftp as before. Upon exiting ftp, Autoexpect would also exit and present them with an Expect script, by default, called "script.exp".

This is graphically shown in the following figure. This mimics the style of the UNIX script command which similarly transparently watches a session and at the end provides a log of the session to the user.

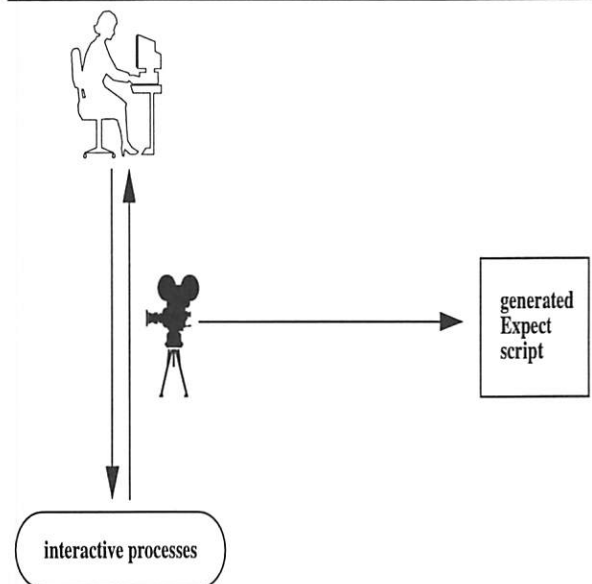


Figure 1: Autoexpect watches an interactive session and generates an Expect script to reproduce it.

Autoexpect can automate multiple interactive applications as easily as a single interactive application. Autoexpect does this exactly the same way that a user does – through the shell. By initially spawning a shell (the default if Autoexpect is called with no arguments), the user can interact, running multiple programs, switching attention from one to another using job control, and other arbitrary interactions. Autoexpect will dutifully note everything and reproduce it faithfully. Job control is supported with no extra effort. For example, the user may press ^Z to suspend a job or ^C to generate an interrupt signal. Autoexpect will generate the appropriate statements to reproduce these so that the interaction can be repeated.

Autoexpect Audience

Autoexpect can be used with zero knowledge of Expect. Autoexpect creates a complete ready-to-run program. Thus, Autoexpect is a good fit for people who know nothing about Expect.

Autoexpect is also useful for users who already have some Expect knowledge. For example, users may want to generalize the resulting scripts, such as by changing repeated sets of statements into loops. Even for Expect experts, Autoexpect is a convenient tool for automating the more mindless parts of an interaction. It is much easier to cut/paste hunks of Autoexpect scripts together than to write them from scratch.

Autoexpect provides a similar value to beginning and intermediate Expect users. An additional benefit is that Autoexpect always provides perfect patterns to match output. Budding Expect users find it useful to run Autoexpect and examine the patterns it has chosen to match program output.

It does not take long to acquire a moderate competency of Expect. Nonetheless, both intermediate and expert Expect users often find themselves in a common scenario: As they are typing something for the second or third time, they start thinking:

“Gee, this could be automated using Expect. But that could take five minutes to write and 10 minutes to debug and I can simply do the interaction itself in two minutes, so I can’t justify stopping and automating it with Expect.”

This is sound reasoning – today. Of course, if the user does the same thing tomorrow and again and again, they soon start to lose time by not automating. Only by taking a step back (“Is this interaction likely to be repeated in the future?”) can the correct choice be made – whether to go ahead and manually interact or to stop and invest the time in automating the interaction.

The shorter and simpler the interaction is, the more likely it is for users to fall into the trap of not considering the advantages of an Expect script. This may seem counterintuitive. In fact, users recognize the value of Expect for long interactions. The more lengthy, painful, or boring the interaction, the more quickly people turn to Expect. But even expert Expect users repeatedly perform interactions that are quick and brief, believing that they’re in a rush and that they can’t afford to stop and automate a problem that they don’t have to think about today.

In short, Autoexpect is worth knowing wherever a user is on the Expect spectrum – from expert to total newcomer.

Potential Pitfalls

It is important to understand that Autoexpect does not guarantee a working script because it necessarily has to guess about certain things – and occasionally it guesses wrong. However, it is usually very easy to identify and fix these problems. The typical problems are:

Timing

By default, Autoexpect produces an interaction designed to run as quickly as possible. However, a surprisingly large number of programs (e.g., `rn`, `ksh`, `zsh`, `telnet`) and devices (e.g., modems) ignore keystrokes that arrive “too quickly” after prompts. If a generated script hangs while waiting for a prompt at one spot, a brief pause may be necessary before the previous command is sent.

Fortunately, these spots are rare. For example, `telnet` ignores characters only after entering its escape sequence. Characters are ignored immediately after connecting to some modems for the first time. A few programs exhibit this behavior all the time but typically have a switch to disable it. For example, `rn`’s `-T` flag disables this behavior (which `rn` refers to as `typeahead`). Listing 1 shows how a script must be written to deal with poorly-designed modems.

Autoexpect supports a “conservative” mode. By enabling this mode, the generated script will pause briefly before sending each character. The pause is not noticeable to humans but is sufficient to pacify sensitive programs. This mode can be enabled all the time or interactively under control of the user.

It is possible to use precise character inter-arrival times to reproduce the original interaction timings. However, this is not desired by most users who want scripts that interact as fast as possible, not at the same speed as humans. Slowing down scripts to human speeds doesn’t guarantee a correct result either since few programs make timing guarantees. Instead, users are at the mercy of the operating system scheduler.

Echoing

Many program echo characters. For example, if a user types “cat” to a shell, what Autoexpect actually sees is:

```
user typed 'c'
```

```
spawn tip modem
expect "Connected"      ;# tip says it has allocated modem
sleep 0.1                ;# pause allows modem to enable UART
send "ATD1234567\r"      ;# otherwise this would be ignored
expect "CONNECT"        ;# and this would hang forever
```

Listing 1: Script for poorly designed modems


```
computer typed 'c',
user typed 'a',
computer typed 'a',
user typed 't',
computer typed 't',
user typed \r
computer typed \r\n
```

A literal rendering into Expect looks like this:

```
send "c"
expect "c"
send "a"
expect "a"
send "t"
expect "t"
send "\r"
expect "\r\n"
```

Without specific knowledge of the program, it is impossible to know if the user is waiting to see each character echoed before typing the next. If Autoexpect sees characters being echoed, it assumes that it can send them all as a group rather than interleaving them the way they originally appeared. Thus, Autoexpect rewrites the fragment above as:

```
send "cat\r"
expect "cat\r\n"
```

This makes the script more pleasant to read. However, it could conceivably be incorrect if the user really had to wait to see each character echoed.

The additional `\n` at the end of the previous expect is added because the terminal driver normally echoes a return character with a return-linefeed sequence. This knowledge is not wired in to Autoexpect. In this situation, Autoexpect merely consumes as many characters as possible until the user resumes typing. This makes scripts very "tight". Rather than three expect commands (one each for the echoed command, the command output, and the subsequent prompt), only one expect command is generated which matches everything.

As an aside, most human-written scripts would not bother with the `"expect \"cat\r\""` either. After all, Expect itself will just skip over extraneous output. Autoexpect could be made to make this optimization except that it raises the concern that the following output could resemble the command. In that case, the generated script would execute the next send before the desired output had actually appeared.

Change

By default, Autoexpect records every character from the interaction in the script. This is desirable because it gives the user the ability to make judgments about what is important and what can be replaced with a pattern match.

On the other hand, the generated scripts are not going to be correct if interactions involve commands

whose output differs from run to run. For example, the "date" command always produces different output. So using the date command while running Autoexpect is a sure way to produce a script that will require editing in order for it to work.

Autoexpect supports a "prompt" mode. In this mode, Autoexpect will only look for the last line of program output – which is usually the prompt. This handles the date problem (see above) and most others.

Style? Not!

Autoexpect scripts will not be mistaken for humanly-generated scripts. Autoexpect uses a variety of features that were always intended for machine-generated scripts. For example, all send commands generated by Autoexpect use the `--` flag, as in:

```
send -- "more"
```

The `--` suppresses any possible interpretation of the next argument as a flag. Obviously, this is redundant here. "more" is not a flag. However, by using the `--`, it is not necessary for Autoexpect to check whether or not "more" is a flag. Thus, for Autoexpect, it is simpler to use the more verbose form. For users, it is simpler to use the less verbose form. There are many operations in Expect that follow this convention. Thus, Autoexpect scripts tend to look somewhat verbose and peculiar.

There are useful things to be gained by studying the output of Autoexpect. For example, it is possible to see how any string must be quoted in order to use it in a Tcl script simply by running Autoexpect and typing the strings in. But in general, Autoexpect should not be held up as a model of human programming style. That is not its goal.

Implementation Notes

Autoexpect is implemented as an Expect script. The Expect script is rather interesting in itself, however discussion of the techniques is beyond the scope of this paper. One relevant item is that Autoexpect includes a hefty template, inserted at the beginning of each script describing things about the generated code – just in case the user does look at the script, they should have a little introduction to Autoexpect's surprising style.

Despite its implementation in pure Expect, the script easily keeps pace with interactive use. For instance, large chunks of output are processed with a single `read()` and appended to an output buffer. Only when the user types a character are the script output buffers flushed. And since users type slowly (relatively speaking), this processing is not noticeable. This is similar to the approach taken in Kibitz, and also in Tk in general in which events are handled entirely using Tcl scripts ([Libes93]).

Todd Richmond (Legato, Inc.) demonstrated the feasibility of automating Expect scripting by

modifying Expect itself. Written in 1991, this experiment was never publicly released, in part merely because it was based on a much earlier version of Expect and no one had the time to integrate it into the current version.

In retrospect, a script-based Autoexpect provides much more flexibility and without loading down the Expect core. Like Tcl and Tk, we should resist changes to the Expect core that can be more profitably accomplished using Tcl itself.

Work In Progress

The publicly-distributed Autoexpect produces straight-line code. This may sound simplistic, but it is valid.

Work is currently in progress on "loop rolling" – the removal of statements by creating loops to produce the equivalent effect. This section describes several example problems that are being studied.

Creation of Counted Loops

Consider the following sequence. It has a repeating send/expect sequence:

```
send A
expect B
send A
expect B
send C
```

This can be rewritten as:

```
for {set i 0} {$i<2} {incr i} {
    send A
    expect B
}
send C
```

This rewrite is not particularly valuable with only two repetitions, but it shows the possibility for handling more repetitions.

Creation of Output-dependent Loops

Consider the following sequence. It has a repeating expect/send sequence:

```
expect B
send A
expect B
send A
expect C
```

If this interaction is repeating until the appearance of C, this is better rewritten as:

```
expect {
    B {
        send A
    }
    C
}
```

If the interaction is not repeating until the appearance of C, this form can still be used although it might be better not to use this form unless there are many repetitions (however "many" is defined). The earlier counted-loop may be more meaningful, however.

Other Problems

These solutions must be generalized. For example, repeated statements may consist of multiple send-expect sequences, not just one. Similarly, nested expect statements must also be supported. For example, consider this interaction:

```
expect B
send C
expect D
send E
expect B
send C
expect D
send E
expect F
```

This could be rewritten:

```
expect {
    B {
        send C
        expect D
        send E
    }
    F
}
```

User interactions occasionally require commands other than send and expect. So these other statements must be handled as well.

Unfortunately, there are no optimal rewriting algorithms. Some script fragments can be rewritten in several ways. Straight-line scripts easily solve the goal of reproduction. The more difficult and less definable goals are appropriateness to a particular task. And what makes good sense for one task may not make much less sense for another. One approach to consider is having a user-driven generator with a graphical interface. The graphical interface would present various ways of "rolling" the code. As the user would select and unselect them, the code would be shown rewritten in the various forms.

Conclusion

For many scripts, Autoexpect saves substantial time over writing scripts by hand. Even Expect experts will find it convenient to use Autoexpect to automate the more mindless parts of interactions. It is much easier to cut/paste hunks of Autoexpect scripts together than to write them from scratch. And beginners may be able to get away with learning nothing more about Expect than how to call Autoexpect.

Acknowledgments and Availability

Thanks to Jeff Bowyer (Institute of Computer Science, Masaryk University, Brno, Czech Republic) and Henry Spencer for assistance in the development of Autoexpect. Thanks to Steve Ray and Jim Fowler for providing suggestions which improved this paper.

Expect and Autoexpect are freely available. However, the author and NIST would appreciate credit if this software, documentation, ideas, or portions of them are used.

The scripts and programs described in this document may be ftp'd as pub/expect/expect.tar.Z from ftp.cme.nist.gov. The software will be mailed to you if you send the mail message "send pub/expect/expect.tar.Z" (without quotes) to library@cme.nist.gov.

Author Information

Don Libes is the creator of Expect as well as the author of its definitive text, *Exploring Expect* (O'Reilly, 1995). Don has written over 80 computer science papers and articles plus two UNIX classics: *Life With UNIX* (Prentice Hall) and *Obfuscated C and Other Mysteries* (Wiley). Don is a computer scientist at the National Institute of Standards and Technology. Reach him electronically as libes@nist.gov.

References

- [Libes93] Libes, D., "Kibitz - Connecting Multiple Interactive Programs Together", *Software - Practice & Experience*, John Wiley & Sons, West Sussex, England, Vol. 23, No. 5, May 1993.
- [Libes95] Libes, D., *Exploring Expect: A Tcl-Based Toolkit for Automating Interactive Programs*, O'Reilly and Associates, January 1995.
- [Ouster] Ousterhout, J. K., *Tcl and the Tk Toolkit*, Addison-Wesley, 1994.

An LPD for the 90s

Mark Fletcher – SAS Institute Inc.

ABSTRACT

lpdx was designed as an enhancement to LPD to meet the demands for seamless inter-platform printing and to simplify administration efforts relating to a large diverse printing environment. With the tremendous increase in the number of printers in the last few years, it is important to have an easy to use, network based administration tool that eliminates rlogins to servers and special accounts. As network traffic increased we noticed numerous timeout and hanging problems with the rlpdaemon/lpsched print servers we were running on our HP-UX systems. lpd's features were brought about by the increase in the number of printers, the downsizing of many printing duties from the mainframe, and the demands an ever increasing PC base presented. Some of these features include form and job class selection, network-wide job control, printer forwarding, and better control over administration access.

Introduction

I have been studying and developing Unix printing for the last few years. In that time it has amazed me that Unix printing has seen such little growth since the original development of LPR/LPD. Perhaps this is because we keep expecting the "paperless office" to materialize; thinking that the printed page will go the way of the typewriter. The fact is, printed output has grown steadily every year and there does not appear to be an end in sight.

Most vendors supply some sort of modified version of LPR/LPD with their systems. Our experience has been that these print systems do not adequately address the demands of large, multi-platform, printer diverse environments. Some of the problems we have discovered are:

Lack of seamlessness

Users are required to know unreasonable amounts of detail about print queue locations, commands, options, etc.

Lack of drivers

Most vendor printing systems support only a simple and narrow set of printer types and capabilities.

Lack of scalability

What seems reasonable for administering five or six printers becomes an administrative nightmare as the printer population grows into the hundreds or more. Administrators are forced to log in on multiple servers and run administrative commands. In addition to keeping track of queue locations, administrators must have login accounts on every server that contains a print queue. The problem quickly becomes unmanageable.

Lack of compatibility and simplicity

Most "out of the box" solutions tend to address specific hardware (e.g., An HP-UX system talking to an HP LaserJet 4SI). Very often vendors

complicate their systems with "features" administrators do not need or want.

Lack of capabilities

Many features available on other systems are just not there or are poorly implemented on most Unix print systems. Because most of these systems still depend on underlying programs developed in the 70s and 80s, it's no small wonder that users are greatly disappointed when they learn of the immaturity in Unix printing.

In 1992 we developed an LPR client program called nlp (see proceedings of LISA VI conference, October 19-23, 1992). With this client, many of the above mentioned problems were solved. nlp gave us a printer database that mapped logical printer names to a <hostname, queue name>. This hid the details of server locations and queues from the user. However, since the server was vendor supplied, we still were limited by the capabilities of the server.

As our network grew we found ourselves facing many new printing challenges. I organize these into three categories.

1. SAS has been steadily downsizing mainframe resources. As we migrated printers from the mainframe to Unix, we soon discovered inadequacies that could not be resolved with the current system. The most important of these was the lack of form support. Mainframe users were accustomed to submitting jobs to specific forms criteria (type of paper, etc.) which was then released by an operator to the printer after the proper form had been loaded on the printer. Since many of these tasks were mission critical (payroll, contracts, etc.) this was a very serious problem.
2. Due to the sharp increase in the size of our printer base (we currently have over 350 printers on 150 servers) we found administration time sky rocketing. Administrators still had to log in on the servers to manage print queues

which meant that they needed accounts on all the servers. In addition, they had to determine at each occurrence which server to log into.

3. As our computing environment has matured and diversified across the globe, the demand for a true client-server printing model has become essential.

Design Strategies

We started with the "time tested" lpd source and enhanced it to conform to the client-server model we needed. We had three basic goals in mind as we developed lpdx.

1. An easy to use, client-server based administration tool that eliminates rlogins to servers and the need for special accounts.
2. A robust, down-right relentless print scheduler that won't ever give up trying to print the jobs in its queues. As the size and breadth of our network increased we noticed numerous time-out and hanging problems with the rlpdaemon/lpsched print servers we were running on our HP-UX systems.
3. Form and job class selection, network-wide job control at the user level, and printer forwarding.

These are needs that emerged due to the increase in the number of printers, downsizing of many printing duties from the mainframe, and the demands presented by an ever increasing PC base.

System V Compatibility

With SAS being primarily a System V shop, many of the features of the System V LPSCHED system were added to lpdx; mostly to take advantage of pre-existing software. The most notable is the printer interface program. Unlike the filters used by LPD, lpdx calls an administrator written program (the interface program) which handles the actual printing of the job in exactly the same manner as LPSCHED does. The lpadmin(1M) man page provides a description of the interface program. lpdx passes "well defined" parameters to the interface program (e.g., job number, login name, title) and it is the responsibility of the interface program to produce a banner page and a complete data stream suitable for the particular printer type for which it is written. Since most of our printers are network attached using network print servers, the interface program serves as a TCP client to the network print server to transfer the data. This eliminates the need for lpdx to handle output devices which

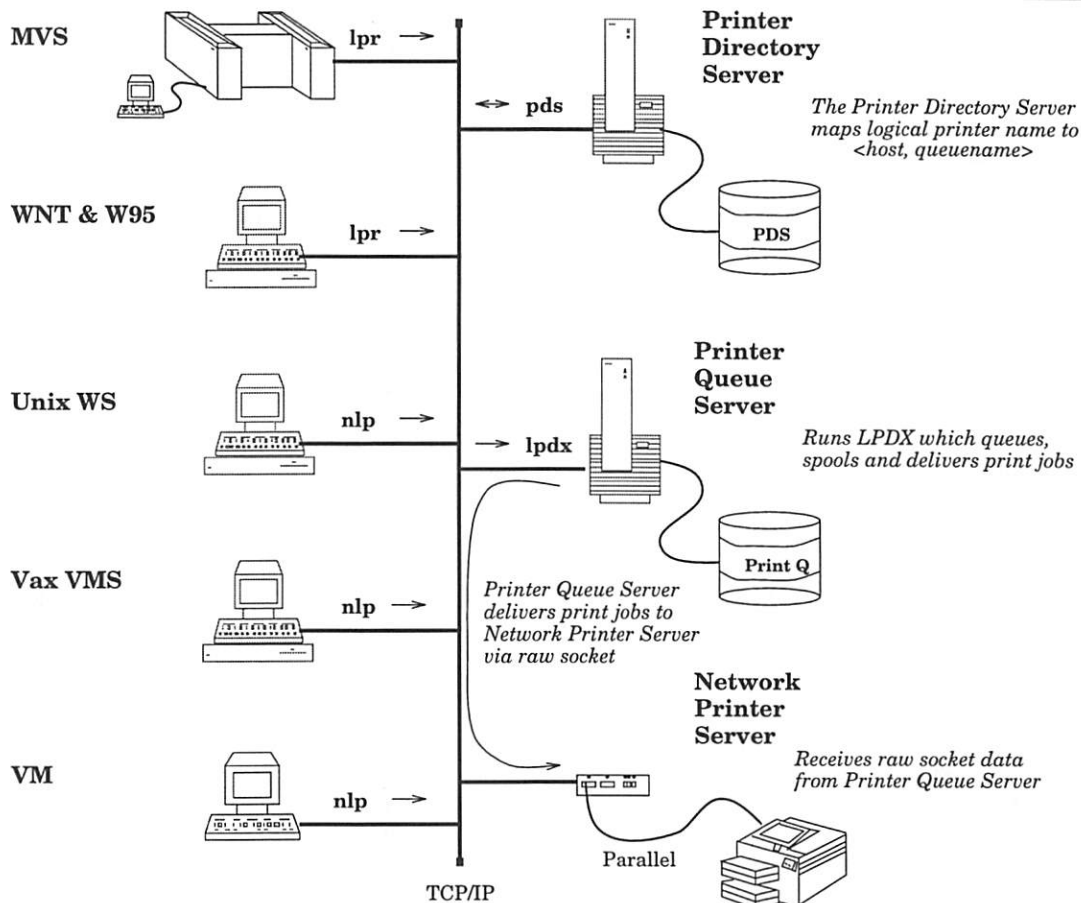


Figure 1: Typical SAS printer configuration

further simplifies the interface program as well as lpdX itself. See Figure 1.

The lpdX Directory Tree

The directory structure of lpdX is a hybrid of BSD's LPD and System V's LPSCHED. We took the features we liked from each and combined them with the new ideas we had to come up with the structure shown in Figure 2. To simplify system administration, all the files and directories relating to lpdX are maintained under a single directory tree. Additionally, we made sure the actual spool directory (request) could be replaced with a symbolic link so that the spool could be easily moved to accommodate disk space considerations. Also, because of the client-server administration tool, there is no longer any need for administrators to be "cd'ing" and "ls'ing" their way around the spool directories. This enabled us to tighten down security on the spool directories (771 or even 711) which eased the minds of those printing confidential information like payroll data and employee records.

The printcap File

The concept of a printcap file is maintained with lpdX. The file itself is moved from /etc to lpdX's *etc* directory. Typically, we only use the *sd* capability but lpdX will honor any of the valid capabilities of LPD including everything that would be needed to create a physical device directly on the server such as the parallel or serial ports.

Use of the rhosts File

The *rhosts* file can be used to restrict specific machines and/or users from accessing the print queues on a particular server. This feature allows individual groups to create "personal" printers that only certain users can access. The syntax of this file is the same as */rhosts*.

RFC 1179 & Extensions

Care was taken to assure that lpdX remain compatible with RFC 1179 which defines the protocol for LPR/LPD. This means that any LPR client will work as well with lpdX as it does with LPD. Figure 3 shows the standard RFC 1179 protocols and the lpdX extensions. A detailed description of the extensions will follow.

A Tricky Start

Starting lpdX is a bit tricky. Figure 3 shows that you send a '\007' to start lpdX. To what do you send this byte since lpdX is not running? We handled this problem with *inetd*. First, we chose a reserved port that was not being used. Amazingly, 516 was available. This was great since it is right next door to 515 (the line printer port). Therefore, we added this line to */etc/services*:

```
startlpd 516/tcp
```

The start command will then open port 516 on the server with which we wish to start lpdX on. The last detail is to add a line to the */etc/inetd.conf* file:

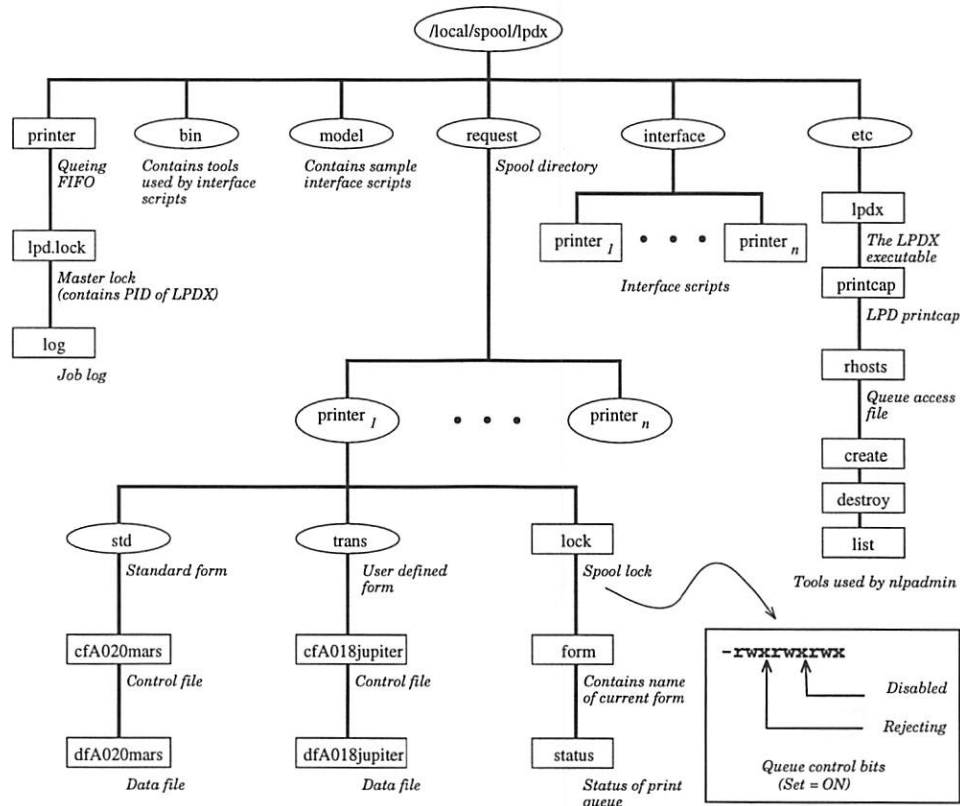


Figure 2: Directory structure of lpdX

```
startlpd stream tcp nowait root ...
/local/spool/lpd/etc/lpd lpd
```

With this, lpd can be started from any machine on the network without having to log in on the server, or even knowing which server it is running on! Stopping lpd is a simple manner of sending a '\010' to lpd which causes it to `exit(0)`.

Disable and Reject

The *disable* and *reject* features of LPDSCHED was something we wanted to be sure and carry forward with lpd. LPD, of course, already supports *disable* so we just had to enhance it slightly to support *reject* which is used to prevent users from submitting jobs (in the event that you don't want jobs piling up on a printer queue that is going to be down for some time). LPD uses the "group" x bit of the *lock* file to disable the queue for the printer. Since that was the only bit manipulation done to this file, we simply followed suit and used the "owner" x bit for our *reject* function (refer to the box in Figure 2). Disabling/Rejecting or Enabling/Accepting are now easily accomplished by toggling the appropriate bits in the printer's *lock* file.

Job Forwarding

Forwarding jobs was another very easy function to add. Since we kept the *printcap* facility completely intact, we simply utilized the **rm** capability to cause anything submitted to the printer to get redirected to a "remote" printer. Administrators simply issue the "forward" command to the printer admin tool which

sends a '\022' to lpd causing lpd to place a **rm** line in the *printcap* file. To stop forwarding, the line is simply removed.

Forms Control

The implementation of forms constituted the most drastic modification to the LPD code. We decided to use directories to hold all jobs of a common form type and to create a form file at the base of the queue directory which contains the name of the current form. This scheme simplifies the printing side of lpd in that it simply looks in the form file then goes to the appropriate directory and prints the jobs it finds there. However, this complicates the receiver side of lpd a bit. When a user specifies a particular form, the form name is placed on the "class" line of the control file. As you can see from the example in Figure 4, this user has specified the form "TRANS" which we will assume means that the user intends for the job to wait until transparencies are mounted on the printer before the job is printed.

```
Hntsys.sas.com
Pmark
JWindows Print Job
CFORM=TRANS
Lmark
TSample print job
fdFA016ntsys.sas.com
UdFA016ntsys.sas.com
NC:\DOC\SAMPLE.PS
```

Figure 4: The control file from a typical LPR client

RFC-1179 protocol

\001queue\n	Print any waiting jobs
\003queue list\n	Send queue state (short)
\004queue list\n	Send queue state (long)
\005queue user-name list\n	Remove jobs
\002queue\n	Enter receive printer job mode
\001\n	Abort job
\002count cfname\n	Receive control file
\003count dfname\n	Receive data file

Extensions to RFC-1179 used by lpd

\007\n	Start scheduler
\010\n	Stop scheduler
\011queue reason\n	Disable queue
\016queue\n	Enable queue
\013src-queue dest-host dest-qname list\n	Move jobs
\006queue device mode\n	Create queue
\014queue\n	Delete queue
\015queue form\n	Change current form
\017logname \n	List log
\020queue\n	Accept queue
\021queue reason\n	Reject queue
\022src-queue dest-host dest-queue\n	Forward queue

Figure 3: RFC-1179 control lines and lpd extensions

When `lpdx` receives jobs and spools them, it can not know which form directory to put the job in because it must open the control file to get that information (Figure 2). We decided to receive all jobs in the “std” directory (the default) and then move them appropriately after fetching the form name from the control file. This way, overall performance is not penalized for an exception. Since most jobs are destined for the “std” directory, they are already in the right place. Only if a job is destined for another form is it moved. The performance hit is not serious since all the directories are on the same file system and the effect is basically a file rename. Also, we decided to exploit the short and long forms of the “job list” command defined in the RFC. When `lpdx` receives a “short job list” command, only jobs in the current form are listed. A “long” command will display all jobs in all forms for that printer. `lpdx` tracks the current form using the *form* file kept at the base of the printer queue directory. The contents of this file simply contains the name of the current form. Administrators can change the current form by sending a “\015” to `lpdx` followed by the name of the desired form and `lpdx` writes this name to the file.

Deleting and Moving Jobs

`lpdx`’s ability to move and delete jobs has been significantly improved over what either LPD or LPSCHED provide.

We decided to relax LPD’s job delete (cancel) rules a bit to allow users with the same username on more than one system to cancel their jobs from any machine. `lpdx` just checks to make sure the user id matches the user id of the job being canceled. It does not pay attention to the name of the client machine like LPD does. We find this works out well at our site but may be too permissive for others.

We went all out on job moving. If you look ahead to Figure 7 you see a very flexible job moving capability. We provide two tools to accomplish job moving. The user command, `nlpmv` allows a user to

perform all the various types of moves to his/her jobs. The `nlpadmin` move allows any and all jobs to be freely moved anywhere throughout the network. `nlpadmin` will be described in a later section.

Job Logging

`lpdx` generates two types of log entries. First, a print job log is kept in `lpdx`’s home directory and contains entries for all the printers defined on that particular server. The detail contained in this job log is dependent on the printer’s *interface script*. In addition to the job log, `lpdx` also writes system type messages and error messages to `/usr/adm/syslog`. Typical messages might include full disk errors or network dropouts. Between these two logs, we can get a pretty good idea about problems that occur.

Listing Jobs

LPD’s job listing output was dramatically changed. We decided to make the “long” listing show the contents of all forms on the printer and the “short” listing just the current form. Figure 5 is a listing of the “long” command from an LPR client.

LPR Clients

`lpdx` is fully compatible with standard LPR clients however the use of the “class” field has been changed to allow print job options to be passed to the printer’s interface script. Example options are things like job orientation, banner page suppression and forms selection. Here is an example of how a form designation can be specified with an `lpr` command:

```
lpr -Phplj2 -Cform=trans foil1.ps
```

As mentioned earlier, `nlp` is a special LPR client that accesses a printer database to translate logical printer names to <host, queue name>. We have had great success mixing LPR clients from lots of different systems with `lpdx`. Our largest growing community is currently Windows NT.

Robustness

The printing side of `lpdx` was beefed up a bit to ensure that every effort is made to find a job that shows up in the queue. To this end, a call gets made to

Lpd v1.1	hplj2,midnight	Current form:std	Model:lj4		

std:					
Rank	Owner	Job	Files	Time	Total Size
1st	bud	477bigiron	B30S/VPSTCPIP/BUD.S	10:09:24	1301
2nd	jim	157panther	fiftyways.txt	10:10:03	1348
trans:					
Rank	Owner	Job	Files	Time	Total Size
1st	mark	157panther	a, b, c	15May96	1295
2nd	martha	19myhomepc	C:/SPOOL/~LP00105.TMP	10:15:15	6768
3rd	sally	201bigiron	B20S/VPSTCPIP/SALLYBJT	10:18:32	1301
4th	mark	182panther	fiftyways.txt	10:25:22	1348
5th	mark	183panther	fiftyways.pcl ...	10:32:12	8561

Figure 5: Listing from an LPR “long” display command

the "printjob" routine following every request lpd receives. If a "list job" request is received, for example, lpd will "bump" the queue after the request is processed. The performance impact to do this is low. If there are no jobs in the queue "printjob" exits after just a few commands. We have not seen a single occurrence of a stalled queue with lpd though we experienced many problems with our old spooler in this regard. Additionally, each time a job arrives, lpd performs a series of "sanity" checks to make sure everything looks healthy. It syslogs any problems it encounters. This has taken a lot of mystery out of our printing environment.

nlpadmin – The Administrative Tool

lpd is entirely managed by a network client called nlpadmin. At no time is it necessary for an administrator to log into the server or even be concerned with which server is in question. nlpadmin is a menu driven tool which issues commands to lpd using extensions to the RFC 1179 protocol for LPR/LPD previously discussed from Figure 3. Figure 6 shows the nlpadmin main menu screen.

Among its capabilities, nlpadmin can start up lpd or shut it down. Queue management includes disable/enable and reject/accept borrowed from the System V lpsched. A very powerful job move command allows jobs to be moved between forms on the same server as well as between servers. Administrators can view job logs and system logs as well as forward jobs from one print queue to another. New print queues can be easily built or deleted using the create and destroy commands. Create and destroy are actually shell scripts in lpd's *etc* directory. These commands can be modified by administrators at any time to meet specific system requirements or special purposes.

Because nlpadmin is just another client to lpd, many instances of nlpadmin can be on an administrator's screen at any given time. Printers anywhere on the network, whether in the same city or even the same country, can be administered simply by changing printer names within nlpadmin.

A summary of the nlpadmin commands is listed in Figure 7.

Securing nlpadmin

Security is handled via a lookup table kept in a disk file. nlpadmin runs *setuid root* and upon invocation it looks up the real user name in the lookup table and checks the user's authorization before allowing the user to proceed. We keep our nlpadmin authorization file in AFS so that it can be accessed by nlpadmin on most of our workstations. We use AFS ACLs to restrict access to this file.

Here is an example of the nlpadmin authorization file:

```
## Administrators of ALL printers
mark      *
jon       *
jim       *
helpdesk  *

## Publications
julie     :hppubs1:hppubs2:
sam       :hppubs1:hppubs2:

## QA
testy     :hpqa1:hpqa2:
picky     :hpqa1:hpqa2:
```

This method of authentication allows us to easily track all our printer administrators. Also, "changes" made to the system by administrators is logged so that a history of all maintenance on a print queue is tracked.

Roll-out

lpd was placed into production at SAS Institute in April 1996. Since that time, our help desk is reporting far less problems with hung jobs and hung queues. We attribute this to the robustness of the lpd algorithms. Administrators have found nlpadmin to cut their diagnostic efforts to a fraction of what it was. They have been able to diagnose problems by reading the logs and have made repairs that they would normally pass on to second level support. Many security holes have been plugged because administrators no longer have to log in to other machines to administer their printers. With the implementation of forms support, many more mainframe tasks have been successfully moved to TCP/IP as well. Plans are underway to convert our European offices to lpd later this year.

NLPADMIN Commands:

start	stop	enable	disable	accept	reject
form	forward	move	list	create	destroy
nlpstat	deljob	printer	adduser	help	quit

Enter help 'command' for syntax.

Current printer:hplj2 Current server:<server2.sas.com,hplj2>

Comm: Net:mpx,lp20.net,pl

nlpadmin(hplj2)#

Figure 6: Main menu of nlpadmin

Futures

Future plans for lpdx enhancements include more integration of SNMP traps into lpdx to monitor printer status and alert users/administrators of stagnant queues, hung printers, etc. We plan to design GUI interfaces to nlpadmin as well as other user interfaces. We also have a project underway to port lpdx to Windows NT for support of dial-in users with locally attached printers. We should take another look at the nlpadmin authorization file as it was placed into AFS out of mostly laziness. The PDS should probably administer this file and be controlled via client-server.

Mail at SAS Institute Inc.; SAS Campus Drive; Cary, NC 27513. Reach him electronically at mark@unx.sas.com.

Author Information

Mark Fletcher has been administering and developing system level tools for Unix systems for the last 10 years. He has been interested in printer system implementations for the last several years and is currently working on printer integration techniques for Windows NT and Unix. Mark can be reached via, U.S.

```

start:    Starts the LPD print spooler on the server for the current printer.
stop:     Stops the LPD print spooler on the server for the current printer.
enable:   Enables the printing of jobs in the printer's queue.
disable:  Stops jobs on the printer from printing.
accept:   Causes the print spooler to resume accepting jobs.
reject:   Causes the print queue to stop accepting jobs.
form:     Change the active (current) form on the printer.
           Syntax: form=new_form
forward:  Causes any jobs queued to printer to be forwarded to another printer.
move:     Moves job(s) to other forms or printers
           Syntax:
             move Destination Source
                   Destination:
                       form=formname
                       destination-printer
                       destination-printer:formname
                   Source:
                       form=formname
                       job(s)
                       user(s)
                       -all

Examples:
           move form=std 207bullet 118daisy
           move hpljl:let sasabc

list:     Lists the printer's job log or system error log.
create:   Creates a new print queue on the server for the printer.
destroy:  Deletes a print queue from the server.
nlpstat:  Displays print queue
deljob:   Cancel jobs.
printer:  Changes the printer nlpadmin is acting on.
adduser:  Edits the nlpadmin user authorization file.
help:     Displays help on a specified command.
quit:     Quits nlpadmin

```

Figure 7: nlpadmin command summary

RUST: Managing Problem Reports and To-Do Lists

Craig Ruefenacht – University of Utah

ABSTRACT

Organizing and managing to-do lists and user reported problems efficiently are important tasks for system administrators. The overhead of organizing them involves time. As the to-do list and problem reports increase, so does the time required to organize and maintain them. Without an effective method of managing these lists, important issues may be ignored and forgotten.

This paper describes our experiences in evaluating and implementing problem and work tracking software. Initially, we examined and tested currently available software. We found, however, that they poorly addressed some problems facing our computing facility. We therefore designed and implemented a ticket system that would best fit the needs of our network.

Defining the Problem

E-mail is a tool commonly used to report problems on a network. In addition, problems are also reported in the hallway, on sticky notes stuck to doors, and over the phone. Without an effective way of managing these problem reports, some of the important problems go unaddressed for long periods of time. The longer a request goes without being looked at, the higher the chance for it to be forgotten. This creates an unhealthy atmosphere for users, staff, and researchers.

The experience we have had is similar to other sites, which are discussed in [1], [2], [3], and [5]. Users of our network did not understand why it took a long time for us to address user reported problems. Some users expressed concern about this latency of problem resolution. Users also expected individual staff members to remember problems discussed in the hall and over the phone. They were unaware of the number of problems and issues facing our facility staff. Besides taking care of user reported problems, our staff maintains and upgrades various software packages, implements changes that reflect current and future growth estimates, installs new hardware and computer labs, and takes care of the administrative items needed to keep a UNIX network running. Without having a system for managing these issues, some of them are overlooked and sometimes end up in a "black hole".

Site Information

The University of Utah College of Engineering and Computer Science computing facility is a heterogeneous network of more than six hundred workstations and fifty servers, running twenty-four different operating systems. These include SunOS (v 4.1.2, 4.1.3, & 4.1.4), Solaris (v 2.3, 2.4, & 2.5), HP-UX (v 9.03, 9.05, & 10.0), Irix (v 5.3, 6.1, & 6.2), AIX (v 3.2 & 4.1), BSD (v 4.3, 4.4, NetBSD, and our own home-grown version), Ultrix v4.3, Linux, Microsoft

Windows NT (v 3.51 & 4.0), Microsoft Windows95, and Macintosh OS. Our network is used by more than forty-five hundred users for tasks ranging from classwork to faculty research. Both NFS and AFS filesystems support our network, providing services and close to a terabyte of disk storage. We have five Fast Ethernet, four FDDI Token Rings, and more than thirty-five ethernet segments connecting the various pieces of our network together.

Many problems arise with this kind of network. Over the past ten months, more than seventy-two hundred problem reports have been submitted to our facility staff, the majority coming from novice users. In addition, more than two hundred work requests have been logged and processed. During the beginning weeks of an academic quarter, the average number of reports received per day can reach seventy-five or more. The facility staff, composed of nine full time system administrators, three full time and two part time hardware technicians, and eight part-time student operators, must handle all these problem reports.

Things Needed to Change

The problem reporting system we were using a year ago was inefficient. Our facility problem reporting system was broken down into five e-mail mailing lists. Each was devoted to a subset of all problems being reported. When users sent e-mail reporting a problem to a mailing list, it was forwarded to the staff members on the particular mailing list. The rule of thumb was that the first staff member to decide to take care of the problem report would be responsible for sending a reply back to the user. They would also be responsible for informing the other staff members that they were working on the problem.

A problem with this setup was that our staff members did not know who was working on which problem reports. Sometimes this caused overlap where two people would work on the same problem at

roughly the same time. It was usually after the fact that both staff members found out about the overlap.

Another problem was having pieces of e-mail from users get lost in the stacks of e-mail from other users. Trying to manage hundreds of messages in your inbox is an art that even experienced system administrators can struggle with. The embarrassment comes a couple of weeks later when the user inquires further about the problem, only to find out that somehow no one knew the problem existed.

These problems were supporting an unhealthy computing environment. It was not providing good rapport between us and the user community. To help combat some of these issues, we began to evaluate ticket systems, which are designed to alleviate the problems we were experiencing.

Ticket Systems

Ticket systems are designed to manage a list of items for a group of people. There are several such systems available for Internet use. They range from complex commercial software that uses relational databases and efficient search engines to simple public domain systems designed around text-based or dbm format databases.

There are two types of ticket systems used in system administration today. One type, commonly known as a trouble ticket system, is used to organize problems reported by the user community of a network. This type of system assumes that each reported problem has a specific solution even if the problem is recurring. Once a problem has been resolved the first time, information about it is stored and can be recalled if the problem manifests itself at a future date.

The other type of ticket system, commonly known as a work tracking system, is used to help maintain to-do and work lists. It is also used to keep track of work being planned or the work currently being done by the system administrators. Work tracking systems are used by system administrators and do not involve the general user community.

Both types of systems work on the same principle. Each request or to-do item, also called a "ticket," is assigned a "ticket" number. These ticket numbers are used to keep track of each request or to-do item. Information about each ticket is stored in a database and can be recalled for future reference.

The Search For a Solution

Several issues need to be addressed to formulate a solution. How can a software system be best implemented to keep track of the large volume of e-mail that report problems? Does it require much overhead to maintain the system and can it function on its own? Is it easy for users to use and can they use it to get feedback about reported problems? Is it secure enough to allow users read access to the database of tickets

without inadvertently allowing modification? Can the system be configured so that it can also handle the to-do and work lists of our staff? Would it be flexible enough that other programs could be interfaced with it at some point, such as financial accounting software?

Existing Systems

We examined several public domain software packages, which are listed in Appendix A. We decided to stay with public domain because our budget does not have much room for the costs of commercial systems. There are a few systems available that have most of the features and abilities we need. But they were financially infeasible for us to implement.

Existing Features

In the public domain software, there were many features common to all the systems we examined. Each system had an interface that used e-mail to create and add information to tickets. This was an important feature for us because most of our problem reporting is done via e-mail.

Some systems are based on specific mail implementations that limit how they can be utilized. The NEARnet Trouble Ticket System, for example, is based on MMDF and currently does not support sendmail implementations. Another system called Queue-mh is a set of scripts based on the MH mail system. This system is discussed in [5]. To use most of the system, one would have to be using MH. This posed a problem for us because our staff uses a wide range of mail programs, including GNU rmail, pine, elm, MH, zmail, and netscape. In order for our staff to use it effectively, we would all have to convert over to MH.

All the systems we evaluated also implemented a database to store information on each ticket. While a couple of the systems use a relational database format, most use plain ASCII text files, with each ticket being stored in its own file. The ASCII databases are easy to maintain and troubleshoot, and can be looked at with standard UNIX utilities. The use of a simple database structure appealed to us because it does not require special interfaces to access stored data.

User feedback implementations are also standard features among the systems under examination. The most common method provided is a user interface. These interfaces allow users to gather information about problem reports submitted by the user. Some systems also send out automatically generated e-mail messages to users who have submitted problem reports.

After looking through what was available, we decided to install a system called REQ, which was written by Remy Evard at Northeastern University in Boston, Massachusetts, and is discussed in [2]. This system was easy to install. It is written to work with sendmail, which is what our mail system is based on. REQ is written in Perl with several C setgid wrappers around the Perl code.

Missing Features

Once REQ was installed, it became apparent that we needed features which REQ did not provide. Most of these features were not apparent to us until we implemented the system.

The key feature missing was support for multiple mailing lists. As mentioned earlier, we had five mailing lists used for problem reporting. We also had a few other lists that could use a management system. In order for us to maintain several mailing lists on REQ, we had to install multiple copies of REQ, one for each mailing list. This setup became burdensome and was not very practical.

Another problem we ran into was that there were too many C wrappers for REQ. When we installed the system for our five mailing lists, we had a set of seven binaries for each mailing list for each architecture. That comes out to be thirty-five binaries. Multiply that by nine architectures in which we installed the system onto, and you come out with three hundred and fifteen binaries. This setup was more complicated than we wanted.

Making It Work

To make REQ work for us, we had to do several modifications to the system. We removed all C wrappers to make maintenance easier. Removing these wrappers also meant that all our staff members needed to have read/write access to the database of tickets so they could add comments and do other operations on the tickets. For a "closed" system in which only our staff was using REQ, this setup worked well.

The biggest modification that needed to be done was to enable one copy of REQ to manage several mailing lists. This involved several changes to the underlying system. A good understanding of the low level structure of REQ was required to do this.

After these modifications were in place, we were able to implement REQ using only one copy to manage our five mailing lists. Other sites using REQ were also trying to implement it for multiple mailing lists. Because of this need, we made our modified version of REQ available to the Internet community. Because our additions had to use much of the existing low level structure, they were not the most efficient. Although our alterations to the system worked, other issues began to surface which would require more modifications.

Along Came RUST

With the REQ system needing more modifications to resolve some issues that came up, the system was becoming more of a hack and less of an efficient system for managing e-mail. Development of REQ was starting to diminish. The other systems we examined also did not have much development currently going on. The hacks we were adding to REQ continued to be limited to the existing code, and it was

becoming necessary to rewrite major parts of it to improve performance for our setup.

Using the results of our examination of ticket systems and our testing of REQ, we decided to formulate our own solution to the problems we were facing. Under the guidance of our facility director, we began to design our own ticket tracking system. The system, called the Requests, Users, and Sys-admin To-do Ticket System, (RUST) is centered around redesigning and improving what other ticket systems have. RUST also has added functionality and features to broaden its application beyond just problem reporting.

The Nitty Gritty

How RUST Works

The basic operations of the RUST System are common among several existing ticket systems. The main tool that RUST employs as its interface is e-mail, based on an underlying sendmail implementation. Tickets are created in the RUST System from e-mail sent to it. Information can be added to existing tickets by sending e-mail to the RUST System with a special "tag" in the subject line.

E-mail is not the only interface to RUST. Operations can also be done via text and GUI based interfaces. This includes creating new tickets, sending e-mail in response to a specified ticket, and updating information. Our staff can also work on the tickets by either e-mail or interactive interfaces. This provides our staff the freedom to use an interface of their choosing.

Multiple Mailing Lists

The key feature of RUST is its ability to manage multiple mailing lists with only one copy of the system installed. This eliminates the need to install a separate copy of the system for each mailing list.

At the top level, RUST handles multiple mailing lists in much the same way as the Majordomo Mailing List Manager as discussed in [4]. Each mailing list is unique and has its own configurations. RUST maintains each mailing list independently of any other mailing list, and provides a mail interface, much like Majordomo, enabling users to request information about specified tickets.

To-do Lists and Work Tracking

Keeping track of problem reports is only part of the solution. Besides taking care of problems, we also do other tasks. Part of the added functionality of the RUST System is the ability to keep track of what our staff members are working on. This includes ongoing projects and to-do items that are in either their planning or implementation stages.

Other problem tracking software can be used for work as well as problem tracking, but those systems make no distinction between the two. A work ticket looks the same as a problem ticket. RUST resolves this problem by defining two types of tickets; user

reported problems and work tickets. RUST can also store financial accounting information and the total time spent on each ticket.

The Path For E-mail

A major concern we were facing as a computing facility staff was the path of e-mail between users and staff members. Users would often send mail to a specific staff member, asking that this or that problem be addressed quickly. Because the e-mail went to only one staff member, the rest of our staff would have no knowledge of the problem. Another concern we had was the situation where a user would send more e-mail concerning a problem to the staff member taking care of the problem, without carbon copying the rest of the staff members.

Removing these variables of sending e-mail was something we felt would be a great help to managing it. The optimum solution would be to have both the users and staff members use the same system for sending e-mail. While expecting all users of the local network to use the same "program" to send mail was not feasible, having the users and staff members both send mail only to one place seemed reasonable.

The RUST System implements a topology of e-mail where our staff members and user community both send mail to one place. To accommodate this, the RUST System keeps track of several pieces of information about each ticket. One of these is a list of e-mail addresses for each ticket, containing all addresses that have ever gotten any e-mail concerning the specified ticket. When e-mail concerning an existing ticket is received, RUST ensures that each address listed for the specified ticket receives a copy of the e-mail.

This topology for e-mail does not remove the inter-staff e-mail that our staff members send each other. We continue to use inter-staff e-mail for discussing various issues and concerns. It is only e-mail that goes between our staff members and the users that goes through the RUST System.

Local User Access

In existing systems, user interfaces are dependent upon having the ticket database files mounted on the machine on which the user uses the interface. To provide some protection, interfaces are either setgid or setuid to a group or user who can read the database.

For better protection, RUST implements a server/client framework using TCP UNIX sockets. This framework is not dependent upon having the database files mounted on all machines on which a user may run the client interface. The client interface makes a TCP connection to the server to retrieve the requested information. The server runs on the machine that the RUST System files are located. All RUST System files are owned by a pseudo user, and the server runs as this pseudo user. To add maximum protection, the server does not modify any files.

Accessibility For Remote Users

In existing software, there are not many provisions made for remote users to get feedback on problem reports they have submitted. The remote user usually does not have any access to the local network except through e-mail and World Wide Web services.

For the RUST System, we are designing a mechanism for remote users to receive feedback and information regarding their submitted problem reports. A remote user will be able to send e-mail to the RUST System on the local network to request information. The RUST System will process the request and serve the requested information. This mechanism is similar to the Majordomo Mailing List Manager, where Majordomo automatically processes requests sent to it via e-mail [4].

We are currently assessing issues concerning limiting remote access in developing a safe implementation. A key issue is how to prevent remote users from finding possible break-in points of the local network by exploiting its insecurities. Remote users should only be able to request information about reports that they themselves have submitted.

Platform Independency and Protection

With a heterogeneous network, achieving as much platform independency as possible is essential. We decided to ensure that all executable code needed for RUST is platform independent. To achieve this, we decided to write RUST in Perl. Doing this gave us the greatest flexibility across platforms and greatly reduced the overhead of maintaining the system.

During the implementation stage of RUST, there were some problems in having everything written in a scripting language. The UNIX operating system environment does not provide a safe mechanism to run programs written in scripting languages as setgid or setuid. Not having the ability to do this meant that any files that staff members needed to have access to would have to be readable by themselves, with a few having to be writable.

If the RUST System were being used in a "closed" environment, where all staff members could be trusted to have read/write access to the RUST System files and database, there would be no problem. However, the RUST System has the potential to be used by more groups than just our facility staff. When expanding the usage of RUST to general user applications, such as mailing lists dealing with software bug reporting, which may have nothing to do with our facility staff, database protection becomes an issue. Putting the users who are on software bug reporting mailing lists and our staff in the same group and enable all of them the same read/write access to all RUST System database files is not wanted.

There are several ways to implement protection to resolve this issue. The method that RUST will use on a permanent basis has not been determined, as

configurations of various types are being looked into. While having everything written in a scripting language is optimal, sometimes it is not feasible.

Currently, RUST employs a single binary wrapper for each platform that is setgid to a UNIX group that has read/write access to all RUST System database files. The wrapper in turn calls interfaces to the RUST System, which are written in Perl. These scripts do internal user authentication checks to ensure that the user running the interface only does modifications to RUST System files that he or she should be able to.

Modularity

One idea that gained support early on in designing and implementing RUST was the idea of modularity. While looking at other ticket systems, we found that most of them were part of the interfaces to the system. If we wanted to redesign the interfaces to the system, it was not a trivial task. Changing how the interface worked invariably affected the rest of the system in some way.

By employing modularity in the design of RUST we were able to implement a system that was interface independent. The support that Perl provides for module functions and files provided a framework in which this could be done. The core of the RUST System is written as Perl modules, being divided into several module files.

Using modularity enabled us to design simple top-level interfaces. They do nothing more than gather information interactively from a person and then make the appropriate calls to module functions. It becomes a trivial task to modify the interfaces to suit the tastes and needs of our staff.

Designing interfaces is not limited in what can be done. Interfaces of any type can be designed without having to know the low level guts of the RUST

System. This also provides a framework in which other software can interface with the RUST System with little effort. Such software packages may include systems that automatically detect problems before they become big problems, accounting software that keeps track of which staff members are doing what, and search engines designed to search through requests for information.

Database Design

There are a variety of database frameworks that were considered for the RUST System database structure. It was decided to implement the database as a set of ASCII text files. Each ticket in the RUST System is associated with its own file. Each file contains the contents of the original ticket. Whenever new information is received, it is appended to the file.

The structure of the database is broken down further to contain two files for each ticket.

Header File

The first of the two files is called a "header" file, which contains the administrative information about each ticket. Such information includes the user who created the ticket, the staff member who is working on the ticket, a list of e-mail addresses who should get a copy of e-mail concerning the ticket, financial accounting information, and what kind of priority the ticket has.

Ticket Logfile

The other file is a log kept for each ticket. This file is in RFC822 format, so any mail program that supports RFC822 mailfiles can read and understand the ticket logfile. This is where all e-mail concerning the ticket, comments added by staff members and users, and operation log messages are stored. New data is appended to the end of the file, so it can also be used as a chronological record of progress.

```
usage: check-q [options]

options you can specify are these:
  --all
  --category <user|work>
  --help
  --owner <owner>
  --printer <printer_name>
  --priority <priority>
  --queue <queue_name>
  --show <num>
  --status <status>
  --subject <subject>
  --type <active|resolved|archived>

<queue_name> can be one of:
  opers unixsys dsl accounts printers modems rust-bugs
```

Figure 1: Usage for check-q

Example Interfaces

User Interfaces

The *check-q* program is a text based interface for use by the general user to retrieve information about problem reports they have submitted. Its usage is summarized in Figure 1. A sample use of *check-q* is shown in Figure 2. It displays a one-line summary of each ticket that matches the query. Another use of *check-q* is to request the entire logfile for a specified ticket. This can be done as shown in Figure 3.

Administrator Interfaces

In addition to the *check-q* interface, our staff has another one called *action*. It is used to perform operations on tickets. Its usage is summarized in Figure 4. For example, if one of our staff members wants to assign a ticket to a fellow staff member, they issue the following command:

```
action --queue rust-bugs --assign \
      johndoe --num 10
```

The above command will assign ticket number 10 in the rust-bugs RUST list to the user johndoe.

```
command -> check-q --queue rust-bugs
```

```
Date: Wed Jul 31 0:35:52 1996
```

```
Query Filters:
```

```
Queue: rust-bugs
Type: active
User: ruefenac
Owner:
Status:
Priority:
Category:
Subject:
```

```
RUST# Owner      Priority  Status Subject
-----
  4   ruefenac normal    open  RUST doesn't parse mail right
```

Figure 2: Using check-q to query

```
usage: action [options]
```

```
options you can specify are these:
```

```
--add_time <hh:mm> (add time to work_hours)
--append user
--assign <user>
--comment
--create
--kill
--merge --num1 <num> --num2 <num> (puts num1 into num2)
--queue <queue_name>
--num <request_number>
--set
    category
    eta
    prio
    status
    subject
    user
    accounting
--send <user>
--show <num>
--value <string>
```

Figure 4: Usage for action

```

command -> check-q --queue rust-bugs --show 4
=====
TICKET INFORMATION
=====
Ticket Queue: rust-bugs
Ticket Number: 4
Ticket Creation: Tue Jun 25 20:32:13 MDT 1996 (835756336)
Ticket Updated: Thu Jun 27 20:10:58 MDT 1996 (835927860)
Ticket Notification:
Ticket Category: work
Ticket Subject: RUST doesn't parse mail right
Ticket Type: active
Ticket User: ruefenac@cs.utah.edu
Ticket Owner:ruefenac
Ticket Status: open
Ticket Priority: normal
Ticket ETA:
=====
TICKET LOGFILE
=====
From ruefenac@cs.utah.edu Tue Jun 25 20:32:12 1996
Date: Tue, 25 Jun 1996 21:32:00 -0500
To: rust-bugs@cs.utah.edu
Subject: RUST doesn't parse mail right

This needs to be fixed...

RUST isn't getting the "From" header right. Take a
look at the debug files and see where the problem is.

-Craig
=====
Date of creation: Tue Jun 25 20:32:13 MDT 1996
From RUST Thu Aug 8 17:24:28 1996 (839546671)
Subject: Comments added

I'm looking at this problem and will have it fixed shortly.

-Craig
=====
Date comments were added: Thu Aug 8 17:24:28 1996
From ruefenac@emily.cs.utah.edu Thu Jun 27 20:10:58 1996
Date: Thu, 27 Jun 1996 20:10:55 -0600
X-Mailer: Z-Mail (3.2.0 06sep94)
To: rust-bugs@cs.utah.edu
Subject: Re: [rust-bugs #4] RUST doesn't parse mail right

%> message from ruefenac
%>
%> This needs to be fixed...
%>
%> RUST isn't getting the "From" header right. Take a
%> look at the debug files and see where the problem is

Fixed...
-----
Craig Ruefenacht ruefenac@cs.utah.edu
-----
=====
Date mail was appended: Thu Jun 27 20:10:58 MDT 1996

```

Figure 3: Using check-q to display entire logfile

GUI Interfaces

GUI interfaces are also provided for both our staff and users that do the same thing as the *check-q* and *action* interfaces. They are written in Perl using the Tk modules to provide the interface to X.

The Results

The results of implementing our own ticket system resolved many issues we were facing and are summarized below:

- All e-mail coming from users which report problems are organized into one system, enabling us to better manage them.
- Our staff knows which problem reports each staff member is working on. Its no longer a guessing game as to who is doing what
- Work and to-do items are managed in the same system as the user reported problems. The work and to-do items appear side by side with the problem reports, enabling our staff to see the amount of work that needs to be done.
- Interface independency was achieved. This allows interfaces to be written without having to modify the low level structure of the system.
- Users are able to see the amount of work that our staff needs to do. This leads to better user understanding as to why the latency of problem resolution is sometimes large.
- The rule of thumb has become "if it isn't in the RUST System, then it isn't a problem". This has removed the problems reported in the hallway and via personal e-mail. The idea takes a little getting used to, as you can't expect the change to be accepted nor implemented overnight.
- Users no longer have to send e-mail to our staff to find out about the status of their reported problems. Interfaces allow the user to obtain this information on-line.

Other benefits resulted from the side affects of this project:

- Other groups at our site saw the RUST System as being useful for software debugging and other mailing lists. We are currently adding six mailing lists which have nothing to do with our facility staff to the RUST System.
- We were able to discontinue the practice of sending a copy of all user e-mail to each staff member on a particular mailing list. RUST processes the mail, logs it, and that is where the e-mail stops. The result was reducing by a large factor the mail that was being sent to staff members.

Future Work

RUST System development is an ongoing project here at the University of Utah. Each component of the system was carefully designed before it was implemented to ensure that the most efficient methods

were used. There will always be development on the RUST System as the needs of our network change.

World Wide Web

Currently, there is no World Wide Web user interface to the RUST System. An interface of this type would not only be useful for local network users, but would also enable remote users to get feedback on their requests. There are many concerns with a World Wide Web interface. The key issue is user authentication so that access can be limited based upon the user requesting information.

Mail Server

Several issues are involved with the mail server. A user mail server is currently under development and different configurations are being tested. Because problem reports and to-do items can provide outside hackers with valuable information, the mail server implementation will include support for data encryption and authentication.

Availability

The RUST System has been designed to work on other sites besides our own. Its application is general enough that it can be used for problem reporting, to-do list and work management, bug reporting, and other similar applications.

Licensing for the RUST System is provided by the GNU Public License and may be freely distributed. It is available via anonymous ftp to ftp.cs.utah.edu in the /pub/rust directory. This is mirrored at the University of Minnesota, ftp.cs.umn.edu, in the /pub/rust directory. There is also a Majordomo mailing list devoted to general discussion of the RUST System. The address of this list is rust@cs.utah.edu. Anyone can subscribe to this list by sending e-mail to majordomo@cs.utah.edu with *subscribe rust* in the body of the e-mail.

Documentation is provided with RUST on all module functions that are available for use in interfaces. Examples are also provided for those who embark on writing their own interfaces. As with text-based interfaces, the same ideas can be applied toward GUI interface development.

Acknowledgments

The development of the RUST system has primarily been done here at the University of Utah. The facility staff has been very helpful in the design and implementation stages of RUST. Aleksandra Kuswik, Kiri Wagstaff, and Michael Chang have also contributed to this project. The information provided from these people have been significant to the RUST System project.

A significant contribution for GUI interfacing has been provided by Shawn Amundson and the Computer Science Department of the University of Minnesota. Other system administrators from the Internet

have also submitted suggestions for making RUST a more robust system. This has enabled the ideas of others to be a part of our design.

Author Information

Craig R. Ruefenacht is a full time student attending the University of Utah where he is working on a bachelor's degree in Computer Science. Craig works for the Computer Science computer facility staff as a part-time operator and systems programmer. His current interest is improving the environment in which system administrators work. You can reach him via US Mail at the University of Utah Department of Computer Science, MEB 3190, Salt Lake City, UT 84112. His e-mail address is ruefenac@cs.utah.edu.

References

- [1] Elizabeth D. Zwicky, "Getting More Work Out of Work Tracking Systems", *Proceedings of the 8th Large Installation Site Administration Conference*, pp 105-110, San Diego, CA 1994
- [2] Remy Evard, "Managing the Ever-Growing To Do List", *Proceedings of the 8th Large Installation Site Administration Conference*, pp 111-116, San Diego, CA 1994
- [3] David Koblas & Paul M. Moriarty, "PITS: A Request Management System", *Proceedings of the 6th Large Installation Site Administration Conference*, pp 197-202, Long Beach, CA, 1992.
- [4] Brent Chapman, "Majordomo: How I Manage 17 Mailing Lists Without Answering "-request" Mail", *Proceedings of the 6th Large Installation Site Administration Conference*, pp 135-143, Long Beach CA, 1992
- [5] Tinsley Galyean, Trent Hein, and Evi Nemeth, "Queue-MH: A Work Queue Management Package for > 3 Ring Circus", *Proceedings of the 4th Large Installation Site Administration Conference*, pp 93-96, Colorado Springs, CO, 1990.

Appendix A

This appendix lists the currently available public domain non-commercial ticket systems we examined. This list is provided in hopes that it may be useful to other sites.

- **GNATS** – Located at prep.ai.mit.edu/pub/gnu/ as gnats-3.2.tar.gz. This is the GNU Problem Report Management System. It is designed for bug reports, but can be used for other applications such as trouble tracking.
- **NEARNET** – Located at ftp.near.net/outgoing/ as nearnnet-ticket-system-v1.3.tar.gz. This ticket system was written at BBN Systems and Technologies. It uses an Informix Relational Database, Embedded-SQL, and MMDF.
- **NETLOG** – Located at ftp.jvnc.net/pub/GES-Other/ as GES-Packages/netlog as netlog-tt-2.4.tar.Z. This system uses standard UNIX tools

and an ASCII database structure to manipulate requests and logs

- **PTS** – Located at ftp.x.org/contrib/applications/pts/ as pts-1.05.tar.gz. Written by Dean Collins, Chris Hyde, Luke Sheneman, Brian Goff and John Hunter, PTS is a system based on X11R5/R6 and the athena widgets.
- **Queue-MH** – Located at ftp.colorado.edu/pub/sysadmin/utilities/ as queuemh.tar.Z. Written by Tinsley Galyean and Trent Hein, this system is a set of scripts designed to interface with the MH mail system on UNIX platforms.
- **REQ** – Located at ftp.ccs.neu.edu/pub/sysadmin/ as req-1.2.7.tar.gz. This system was written by Remy Evard at Northeastern University. It is written mostly in Perl, using some C programs as wrappers. The database is ASCII based, and there is a tcl/tk and emacs interface for it.

Renumbering: Threat or Menace?

Eliot Lear, Jennifer Katinsky, Jeff Coffin, & Diane Tharp
– Silicon Graphics, Inc.

ABSTRACT

Remember when wide area networks consisted of private lines between a few offices routed by a good old fashion classful distance vector protocol? Ever take one of those networks and grow it by several orders of magnitude? Guess what happens? Things begin to break, and network analysts run scared. Herein, we tell the story of how Silicon Graphics, Inc. replaced our old creaking network design for 26 sites with a new bells and whistles design for 200+ sites. The astute reader can savor the irony of some of the (sometimes painful) lessons we learned along the way.

Environment

Silicon Graphics, Inc. (SGI) is a leading computer manufacturer which employs over 10,000 people world wide. It's the "world wide" part that causes us to have a wide area network (WAN) connecting some 200 sites, including our headquarters in Mountain View, CA.

In its early days (circa 1991), SGI did networks just like most IP-savvy companies. We strung a bunch of 56kbs leased lines and connected them to Cisco routers configured to use IGRP, Cisco's proprietary routing protocol. We expected to grow at a reasonable pace of 20% per year. Conventional wisdom held that we should use a Class B network subnetted for the maximum number of hosts any site would have, in SGI's case 126 hosts per network (/25 in CIDR parlance)¹.

Too bad for the network, the market decided it liked what SGI was doing. The network grew just a bit faster than planned, at an annual rate of 100% during 1992 and 1993. By the middle of 1995 we had over 150 sites connected through a Frame Relay fabric. During this growth spurt, we paid little attention to the routing architecture or addressing scheme.

Problems We Addressed

By the beginning of 1995, several problems were glaringly apparent:

- Routing overhead was quite high. A flapping ethernet interface on a router brought down the entire WAN. Even if everything was stable, there was still a fair amount of routing information transmitted to every router to indicate that nothing had changed.

¹network-number/X indicates that there are X number of significant bits in a route. Legal values of X are 1 through 32, excluding 31, where 32 is a host route. For example: Class A = /8, Class B = /16, Class C = /24.

Copyright © 1996 Silicon Graphics, Inc. – ALL RIGHTS RESERVED

- We were nearly out of network numbers. Because we had allocated networks based on 126 hosts per network, as opposed to the number of hosts they actually had (closer to 20 on average), we were using less than 10% of available address space spread across over 300 networks. At a time when Powers That Be were concerned about depleting free address space, this made it difficult for us to ask for more.
- It was nearly impossible for us to isolate Frame Relay errors. In the initial implementation, Cisco made the Frame Relay fabric appear as a multipoint network, such as an ethernet. Because the fabric was represented as a multipoint network, it was difficult to determine which sources and destinations were experiencing problems when they arose.
- We were running obsolete router code. Some routers were running code as old as 3 years, which is an eternity in wide area networking, leading to inadvertent interoperability experiments. Worse yet, one couldn't count on a particular feature being on any given router.
- We had no configuration management scheme. None of the router configuration files were kept in a central location, and people routinely tripped over each others' work.
- We did not have a reliable way to connect our wide area routers to our headquarters network. We did not even make a distinction between WAN and LAN routers. This proves to make routing quite difficult when the two are using different routing methods.

Fixing Frame Relay

SGI went to a Frame Relay network for several reasons – line cost, ability to have multiple circuits through a single local loop, and reduced lead time for new circuits at existing sites. Because we entered the frame game relatively early, we had to use the early implementation supplied by Cisco. The "multipoint" model made the WAN sites appear as if they were all on a single network. This had the advantage of

allowing us to use a single network number (like 155.11.201.0/25) for all sites.

There's just one small problem. Frame Relay does not behave like an ethernet, it is not a broadcast media. This pseudo-network consists of many point to point links, usually lacking the full mesh that an ethernet provides. This limitation required us to commit a form of routing heresy, as we had to turn off split horizons, a safety feature that prevents routing information from being transmitted on the interface on which it was received. Worse yet, ethernet snooping tools could not be used to locate problems since packets aren't really broadcast.

The simplest solution would be to find a way to get back to point to point interfaces, a model that works well with IP. If you can have a virtual circuit, why not a virtual interface? With virtual interfaces, if a specific PVC² is down, the interface appears down, there are interface errors associated with the PVC, and the split horizons rule can be used. In other words, it looks and smells a lot more like the old serial interface model.

In order to move to virtual interfaces, however, we either had to run the virtual interfaces unnumbered on each end, or we had to assign an entire /25 network. In the long term neither solution was acceptable. Given our address shortage, and since we were attempting to move closer to a "classical" IP network model we were well advised to use the unnumbered interfaces as a transition tool to a more permanent solution. The permanent solution, in which all interfaces are numbered, would require classless IP addressing.

Moving to New Routing

With old fashioned IGRP, each WAN site would see a full routing update every 30 seconds or whenever link states changed. If a SLIP connection came up in Denver, Paris saw an update. Although this was flexible, we nearly broke our backs on overhead. Things could get really bad when an interface flapped, since every site in the world would constantly see routing updates³.

Newer routing protocols associate each network with a network mask, allowing aggregation of a bunch of contiguous networks. For instance, outside an administrative area, one could represent the 64 network routes 169.238.0.0/24 through 169.238.63.0/24 with a single route of 169.238.0.0/18. A router outside of an administrative area would only see a small

number of aggregated routes. In our example, if the Denver SLIP connection comes up, Paris no longer notices, since the SLIP network is part of a larger aggregated route [1]. Clearly we needed to move to an aggregate routing system. This change alone, however, would require every site on the WAN to renumber to new IP addresses.

Additionally, through the use of a modern routing protocol, we would be able to reduce ambient overhead by moving to incremental updates, so that changes are transmitted only when they occur, and stable information is kept off the wire.

The protocol we chose, OSPF, supports classless IP addressing, route aggregation, and incremental updates. We could have chosen Cisco's EIGRP, but we did not want to be tied to a specific vendor's protocol. At the time RIP version 2 was not yet available.

OSPF groups sites into areas. Only sites within the same area should be directly connected to each other. Only routers connected to area 0 (the backbone) may be connected to other areas. This encourages strong meshed connectivity between sites within each area (including the area 0 backbone), but discourages connections between random sites in different areas. Each area within our routing system had a hub router which connects to other hubs and to our headquarters. In addition, each site connects to our headquarters in Mountain View on a separate router from the area hub. We thus extend every OSPF area back to Mountain View. For North America, we implemented this plan using eight non-backbone areas.

Doling Out the Bits

Having decided to move to OSPF, we would have a routing protocol that supports classless IP addressing. This means that we could allocate smaller subnets for sites that had small numbers of hosts. It also meant that we could assign networks with room for only two hosts for our PVCs.

If everyone were able to understand classless IP addressing, then each subnet at each site could be perfectly tailored for the number of hosts it would contain. Of course, we don't live in a perfect world. We live in a world in which there are many different types of hardware running many different types of software, most of which do not understand classless addressing. We had two ways to address our imperfections – buy all new hardware and upgrade to all new software or use fixed subnet masks and a default route within each WAN site. The method chosen by our budget department is an exercise left to the reader.

The fixed subnet mask within the site required us to allocate each network for a site based on the largest network within that site, a far shot better than the largest subnet mask within the entire network, since most of our sites have less than 32 hosts. To determine the size of each subnet at a site we applied the following equation:

²Permanent Virtual Circuit. Generally, a virtual circuit that is permanently established. PVCs save bandwidth associated with circuit establishment and tear down in those situations where certain virtual circuits must exist all the time.

³IGRP can be configured in many ways. We diverged from its default configuration because we needed higher convergence times for transient networks, such as demand SLIP or PPP.

$$netsize = 2^{\text{int}(\log(\text{host} \times G)) + 1}$$

where G is a growth factor. This method allows for backward compatibility at the cost of efficiency. Since efficiency was not our prime motivation we found this to be an acceptable cost. We calculated the number of networks for each site as the maximum of the number of nets multiplied by the growth factor or the number of networks needed to fit the number of hosts.

The annual rate at which new field sites opened was leveling off from 100% to a mere 50%. We assumed that as SGI grew in popularity, the number of people required to satisfy sales demands within a region would increase. However, different regions grow at different rates. Therefore, we needed a flexible allocation plan. OSPF's aggregation between areas gave us such a plan rather naturally.

Once we decided to use classless addressing, we had to choose between one class B or a block of 256 class C addresses. With over 3000 hosts and an entirely separate routing system, our Mountain View headquarters would not use classless routing for the foreseeable future. Since each SGI workstation maintains a unaggregated routing table, the effects of 256 routes would be aesthetically unappealing at the very least, and a performance problem in our older operating systems. But use of a class B would risk incompatibilities within the WAN for those sites that could not take a default route. Fortunately, we were able to use default in every location, and thus the class B.

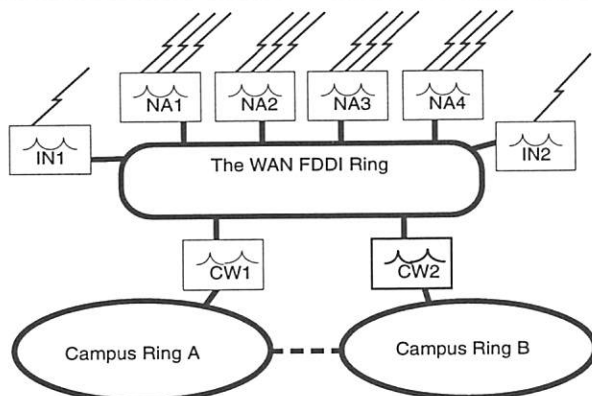


Figure 1: Getting to and from the WAN

Rationalizing Headquarters Access

When we started, we had two WAN routers that came into the campus, each had configuration files over 400 lines. Interaction between campus and the WAN was poorly defined, and interaction between the two routers themselves was not well handled. In order to keep the old routing system in one piece we ran a Frame Relay PVC between the two routers. When that failed, routing would still work, but one might end up going through Boston only to come back to Mountain View on the way to Switzerland. In order to remedy the situation we employed some technology that had matured since the last time we reviewed WAN design-

FDDI. Figure 1 shows how our WAN campus interface appears.

Each router is a Cisco 4700. The ring to ring routers (CW1 and CW2) do not attach to any serial lines. The serial routers (NA1, NA2, NA3, NA4, IN1, and IN2) do not participate in any campus routing (a separate routing system). These two constraints keep the configuration files and the routing far more manageable. Each ring to ring router introduces a default route into the WAN. Thus, if one fails, the other will within a short time provide full redundancy.

Because we were moving toward an OSPF area topology we re-aligned connectivity such that all PVCs within an area came back to one of the serial routers in Mountain View, as well as a regional hub. The hub itself would also come back to a different serial router in Mountain View. This topology was designed for redundancy. In case a link directly connected to the Mountain View serial router failed, the regional hub would provide an alternate path.

Standardizing Router Software Releases

By virtue of their growing complexity, IP routing advances occur on virtually a daily basis. Cisco follows a model where they attempt to get those advances to the customer as rapidly as possible. We call this "The Release of the Week Club". There are usually two major software releases per year, along with numerous subreleases, and bug fixes to boot. Although having short release cycles can be helpful and reflects the pace of the market, different versions behave differently and have different features. With Cisco this is particularly true for OSPF and Frame Relay. When we started the project we had routers running over a dozen versions of the software ranging from version 8.2(3), released in 1990, all the way to 10.3(2), released in 1995. To top everything off we were running brand new hardware.

We wanted to pick with Cisco a release of the software that would both support our hardware platform and also meet our software stability needs.

	10.0(8)	10.2(8)	10.3(7)
Frame Relay	X	X	X
Improved OSPF		X	X
4700 support			X

Figure 2: Determining Releases

The Costs of Renumbering

It's one thing to plan up a storm. It's quite another to have it actually rain. The importance of getting management commitment in a large organization should not be underestimated. We found that the cost of renumbering our WAN was a half day of down time at each site. To mitigate the inconvenience we attempted to align downtime with other events, after hours, or on weekends.

The costs of moving to the new architecture were not limited to site downtime. Indeed classless addressing is a new paradigm, and every network support person needs to understand those changes. Earlier at SGI it was considered unacceptable for even a fixed subnet mask to appear outside of small areas of our network with any mask that did not land on an octet boundary. Our network operations staff needed to be trained to process calls where the local network mask within a particular WAN site might be set incorrectly, or not at all. We supplied everyone with a conversion chart for network masks; bit boundaries; and a table listing each site, assigned network range, netmask, and OSPF area. Even so, it takes time and training for people to understand the new paradigm.

Tools

To accomplish the task of renumbering over 7000 hosts at 83 sites, the system administrator's principle of least work was foremost on our minds. This held especially true when applied to each site administrator whom we would ask to do work. Therefore, we developed a set of tools to assist us.

Doling Out The Bits (Reprise)

Practically speaking no one in their right mind would assign 83 sites classless network address space without some sort of automation. Although usually reserved for MBAs and other budgetary masters, a spreadsheet comes in very handy. Given the number of hosts and networks at each site and a growth factor, a small number of equations can determine the address space needed for each site. If that list is sorted (a common spread sheet function) based on demand within areas, the actual address assignments can then be determined. This also requires that the areas be sorted based on demand.

The resultant spreadsheet can be used to generate tables of sites, their address assignments, and their

network masks for both computer and human consumption. Put simply, without the darn thing you're probably dead in the water.

Renumbering

The script used at each site did as much as it reasonably could to automate the task of renumbering. Mondo, as we called it, took as input the old /etc/hosts file for each site, and produced as output a new hosts file that used the new assignments. In essence, Mondo is a poor man's version of DHCP [2]. Renumbering needs, such as those described in this document or others as related to Internet connectivity, present a strong argument for the use of DHCP. While later versions of SGI's operating systems support DHCP, few if any workstations within the company used it. Still, this was only part of Mondo's responsibility.

Once the hosts file was generated, the script would then push a reconfiguration script to each host within the domain. This script would run when a host was shutdown⁴. It created a new hosts file, created a new resolv.conf, modified the NVRAM configuration, set the appropriate subnet mask for the site, and installed a new SOCKS proxy configuration file that includes the new network [3]⁵.

Not only did we have to document everything we were doing at each site, but we needed to leave

⁴Our extremely explicit instructions did not include the warning that machines should not simply be unplugged or otherwise reset without going through the normal shutdown procedure. This nailed us in at least one instance.

⁵SOCKS is used for Internet access. SGI runs an older version that requires enumeration of every internal network. While one might think that such a file shouldn't change over time, when it does it has to be updated everywhere. The newer mechanisms for such services that do not use network numbers, such as the one used by the Netscape Navigator, are a welcome relief to this problem.

```
mondo now running on momserv.denver in denver.sgi.com.
WAN Site: Denver
New IP Range: 169.238.64.0 to 169.238.72.255
Subnet mask: 255.255.255.128
Found the following networks:
    155.11.155.0
    155.11.155.128
You have 18 new networks.
Replacing 155.11.155.0 with 169.238.64.0
Replacing 155.11.155.128 with 169.238.64.128
writing out new hosts file into /etc/hosts.new.
building package
pushing packages
cannot rcp to napili.denver.sgi.com
You will need to renumber the following hosts by hand:
    napili.denver.sgi.com
```

Figure 3: Mondo

reminders in prominent places, both for our network operations staff and for the site support staff, who in some cases might be a sales person or a graphics engineer (i.e., someone whose expertise is not system administration). We listed in each domain host file the subnet mask for each network used, the first and last valid host addresses, the reserved and broadcast addresses. In addition, we prominently placed a site's subnet mask at the top of each host file.

Mondo could only renumber SGI computers which do make up the vast majority of our network. Even so, Mondo was always billed as a system administration tool and not a replacement for the system administrator. Every site that was renumbered had some sort of small quirk, such as non-SGI hardware, where renumbering needed to take place by hand.

Within a network the size of SGI's the benefit of corporate standards quickly becomes clear. We renumbered sites ranging from two to over two hundred hosts. The time it took to renumber small sites was usually close to, and sometimes exceeded, the time it took to renumber large sites. Large sites tended to have a full time system administrator who understood the value of sticking to corporate procedures. Conversely, small sites did not see the value, and in some cases didn't even know of the existence, of those standards. When attempting to write the procedures and the scripts to be run by WAN sites, we aimed for a 99% solution. In a network of 7,000 hosts that meant we would have 70 exceptions that would need to be manually configured. We probably came in closer to 97%, handling around 200 exceptions.

DNS

Each site is responsible for generating DNS records for its computers. SGI uses an custom-written tool which takes as input the /etc/hosts file and a

configuration file. The tool generates both forward and reverse DNS maps, as well as a named.boot file, with properly updated SOA records. We modified this tool so that it would output proper reverse maps for classless addresses [4].

We also wrote a program that would take as input a configuration file and generate the records necessary to delegate domains and networks in our root DNS. For instance, the octet triple 169.238.100 could have addresses 0-127 allocated to one domain, 128 through 191 to another, and 192 through 255 to yet a third.

The Instructions Generator

At different times we needed to generate specific instructions for each site. We wrote a mail merge program which took spreadsheet output as input, and mailed to each site administrator a set of instructions with nifty customizations such as the site router name, address, and its model. This tool helped us upgrade 83 routers in five days with minimal downtime.

The Router Configuration Checker

At some point the complexity of a routing system exceeds the understanding of enough people such that people begin to feel really nervous. This is made plain when one moves to a new routing system. NET-SYS Technologies has created a product that simulates a WAN. One hands it router configuration files and it points out inconsistencies and errors within the entire routing system. This thing even knows about Cisco bugs within various releases⁶.

⁶Sort of reminds one of most VT100 emulators.

```
# This hosts file was generated on Sun Mar 10 18:26:42 EST 1996 by
# mondo version 1.1.
#
#
# This is the hosts database for clubfed.sgi.com. It should
# only be used in its entirety on the relay system
# beyond.clubfed.sgi.com.
#
#
# The subnet mask for this domain is 255.255.255.128.
#
# The following networks are used in this domain:
#
#   Network           First Address   Last Address   Broadcast
# 169.238.1.0         169.238.1.1   169.238.1.126 169.238.1.127
# 169.238.1.128       169.238.1.129 169.238.1.254 169.238.1.255
# 169.238.2.0         169.238.2.1   169.238.2.126 169.238.2.127
# 169.238.2.128       169.238.2.129 169.238.2.254 169.238.2.255
```

Figure 4: An Example /etc/hosts file

Lessons Learned

We renumbered every SGI WAN site in North America in a period of six weeks. It took us six months to plan the activity, and cost us a large amount of people's time. The task would have been made easier had we previously deployed DHCP. We forgive ourselves in this case as in others, as technology such as DHCP is still emerging. The same logic holds true for the old routing and addressing schemes we used way back when. In 1991, OSPF and classless addressing were not available to us.

We also must emphasize that the reason we renumbered was that we wanted to protect our routing system from excessive storms, while still quickly converging. We did not renumber, as many sites do, today, because our Internet provider asked us to. Similarly, we could have chosen to use private address space. Because SGI communicates with many companies and tends to buy others, we chose to not use number that could possibly be used elsewhere, to reduce future complications [5].

Our choice of routing protocol is a continuing point of discussion. As mentioned earlier, we could have used EIGRP to largely accomplish the same ends. We could also have used BGP in certain places. The main reason we did not consider RIP version 2 was that it was not available on the Cisco routers at the time of this project. No matter what routing protocol, the requirement to renumber persists, and the principles used to renumber persist. We chose to address our routing problems through route aggregation, and we merely used OSPF as the means to that end.

In fact, OSPF has some limitations involving its interaction with other routing processes on Ciscos. Until the protocol is extended to handle "not so stubby areas" some routes advertised into OSPF are not aggregatable. Which means routing tables are not as manageable as we like them to be.

The importance of documentation cannot be overstated. The process we used to renumber North America was used to renumber our European network during the months of May and June. Where we documented things properly (especially the parts that caused us problems) our European counterparts were able to sail through.

The use of a test lab was critical, as were pilot implementations for each stage of our network upgrade. Before attempting to monkey with 7,000 workstations, the reader is encouraged to try a smaller number, like twelve.

Acknowledgments

This project was accomplished by a team, of which each of us were merely members. Along side the authors, the team consisted of John Parisi, Doug Herfurth, Wayne Silver, Gabriel Villalobos, Greg

McGrath, Mark Mellis, Oliver Enzmann, and Chan Wilson. Many thanks to our network operations staff for their support and understanding. Numerous people at Cisco and Sprint made each transition as painless as possible. Finally, without the vast experience and boundless energy of Rod Scott this project would not have been possible. In addition, Ed Cole's and Mel Pleasant's superb project management skills as well as Rob Cowan's patience, it's not clear that we ever would have gotten the project off the ground.

Author Information

The authors are members of the Corporate Information Network Services of Silicon Graphics, Inc. and can be reached via US Mail at Silicon Graphics, Inc., 2011 N. Shoreline Blvd., MailStop 730, Mountain View, CA 94043-1389. Jennifer Katinsky can be reached via electronic mail at ariana@sgi.com. Eliot Lear can be reached via electronic mail at lear@sgi.com.

References

- [1] Fuller, V., Li, T., Yu, J., and K., BARRNET Varadhan, "Supernetting: an Address Assignment and Aggregation Strategy", *RFC 1338*, BARRNET, Cisco, Merit, OARnet, June 1992.
- [2] Droms, R., "Dynamic Host Configuration Protocol", *RFC 1531*, Bucknell University, October 1993.
- [3] Koblas, D., Koblas, M., "SOCKS", *USENIX, UNIX Security III Symposium Proceedings*, pp. 77-83, September, 1992.
- [4] H. Eidnes, G. de Groot, "Classless in-addr.arpa delegation", Internet Draft, SINTEF RUNIT/RIPE NCC, January, 1996.
- [5] Y. Rekhter, R. Moskowitz, D. Karrenberg, G. de Groot, E. Lear, "Address Allocation for Private Internets", 2/29/1996.

OC3MON: Flexible, Affordable, High Performance Statistics Collection

Joel Apisdorf, *k claffy* (NLNR), Kevin Thompson, & Rick Wilder – MCI/vBNS

ABSTRACT

The Internet is rapidly growing in number of users, traffic levels, and topological complexity. At the same time it is increasingly driven by economic competition. These developments render it more difficult, and yet more critical, to characterize network usage and workload trends, and point to the need for a high performance monitoring system that can provide workload data to Internet users and administrators. To ensure the practicality of using the monitor at variety of locations, implementation on low cost, commodity hardware is a necessity.

Part I: Design and Implementation

Introduction

In its role as the network service provider for NSF's vBNS (very high speed Backbone Network Service) project, MCI has undertaken the development of an OC3 based monitor to meet these needs. We will describe and demonstrate our current prototype. The goal of the project is to specifically accommodate three incompatible trends:

- Current widely used statistics gathering tools, largely FDDI and Ethernet based, are running out of gas, so scaling to higher speeds is difficult
- ATM trunks at OC3c are increasingly used for high volume backbone trunks and interconnects
- Detailed flow based analysis is important to understanding usage patterns and growth trends, but such analysis is not possible with the data that can be obtained directly from today's routers and switches

Specific design goals that led to the current prototype are

- A flexible data collection and analysis implementation that can be modified as we codify and refine our understanding of the desired statistics
- Low cost, in order to facilitate widespread deployment

The project schedule calls for deploying the monitor in third quarter 1996 in the vBNS. As soon as we

demonstrate its stability, we will make the software freely available to others for use elsewhere. Both the flow analysis code and monitor architecture will be public domain.

Description of the OC3 Monitor

Hardware

OC3MON is an IBM personal computer clone with 128 MB of main memory, a 166 MHZ Intel Pentium processor, an Ethernet interface, two ATM interface cards, and a 33 MHz 32-bit-wide PCI bus. Our first implementation used ATM interface cards built around Texas Instrument's SAR (segmentation and reassembly) chips due to early availability and low cost. The current version of OC3MON uses a Fore Systems ATM network interface card (NIC) for the PCI bus. The Intel i960 processor on this card allows us to optimize OC3MON operation with custom firmware. We made arrangements with Fore to obtain the necessary source code and freely distribute the custom firmware executables along with the source code developed for the OC3MON system processor.

We attach the OC3MON ATM NICs to an OC3 fiber pair carrying IP traffic, connecting the receive port of each ATM card to the monitor port of an optical splitter, which carries 5% of the light from each fiber to the receive port of one NIC. (The dual splitter cost is about \$800; the NICs run about \$1200.) Attached to an OC3 trunk terminated on a switching device (e.g., ATM switch or router), one of the OC3MON NICs sees all traffic received by the switching device and the other NIC sees all traffic transmitted by the switching device. In the vBNS, we will attach an OC3MON to each connection from the wide area ATM backbone to the primary nodes at the super-computer centers (see Figure 1).

Software: Why We Didn't Choose Unix

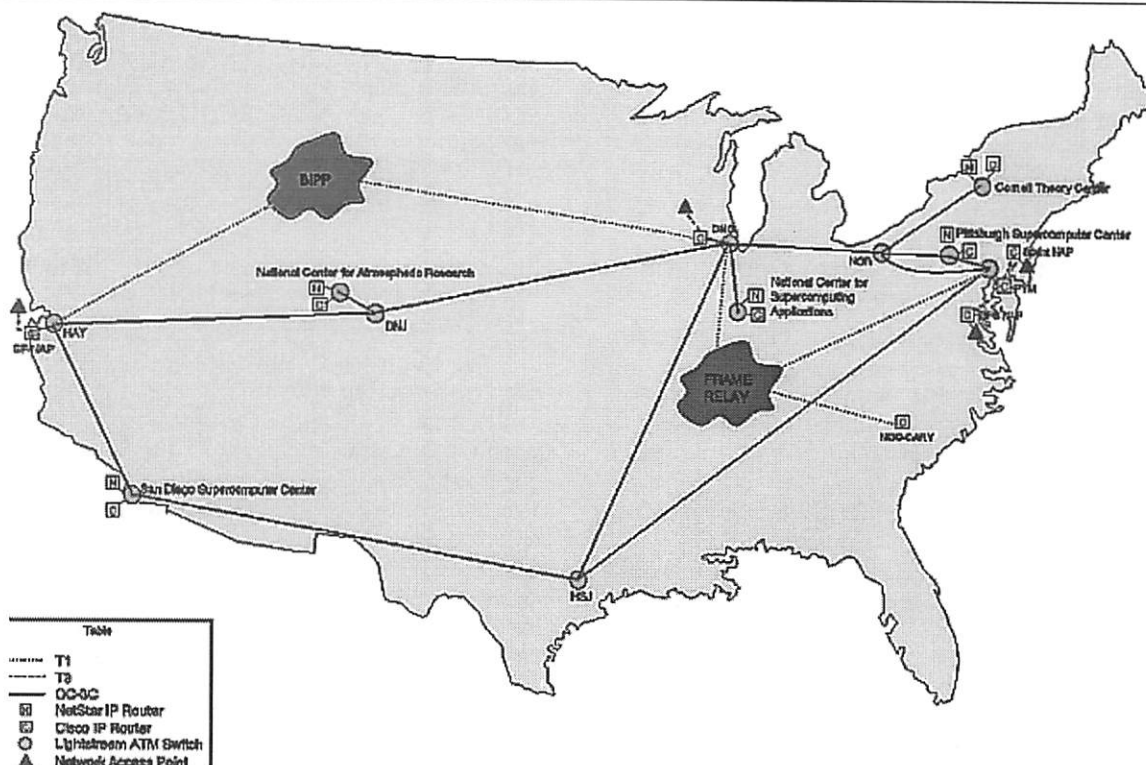
The DOS-based software running on the host PC consists of device drivers and a TCP/IP stack combined into a single executable; higher level software performs the real-time flow analysis. Several design

This material is based on work sponsored by the National Science Foundation, grants NCR-9415666 and NCR-9321047. The very high speed Backbone Network Service (vBNS) project is managed and coordinated by MCI Communications Corporation under sponsorship of the National Science Foundation. The Government has certain rights to this material. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Second, we needed the ability to monopolize the entire machine, which is easier with DOS than Unix. OC3MON needs to provide the hardware with large blocks of contiguous physical memory, so we did not want the operating system to have to maintain knowledge about the memory and possibly fragment, resulting in lower efficiency on card-to-host buffer transfers. We did not want the kernel to suddenly decide it

The disadvantage is that DOS only has blocking I/O routines, whereas Unix would provide non-blocking I/O. But in experimenting with our own workstations, we did not see evidence that Unix single disk I/O was significantly faster than that which we were going to implement. We do hope to eventually port OC3MON to a Unix platform, most likely Linux on a PC.

The software directs each ATM NIC to perform AAL5 (ATM Adaptation Layer) reassembly on a specified range of virtual circuit and virtual path identifiers (VCI/VPI). Note that Cisco routers and Fore switches also support AAL3/4, but MCI does not use it on either the vBNS or their commodity infrastructure because it consumes an additional four bytes from each cell (above the five already used for the ATM header) to support submultiplexed channels within a given VP/VC. Since the LLC/SNAP 8-byte per-frame header that the routers insert already includes a 2-byte ethertype field that allows, if needed, multiplexing of



1996 LISA X – September 29-October 4, 1996 – Chicago, IL

different protocols (IP, IBM SNA, Novell IPX, etc) on the same VC, including AAL3/4 support in the design would not have been beneficial. (In fact the other six bytes of the LLC/SNAP header, for which we have no use, take up so much space in the first cell that even for simple TCP ACKs they squeeze out the 8-byte AAL5 trailer, which then requires its own cell.)

AAL5 makes use of a user-defined single-bit field in the ATM header to indicate whether a cell is the last in a frame. AAL5 also assumes that cells for a given frame will not be interspersed with cells for another frame within the same VP/VC pair. Combined with a single bit of state per VP/VC pair maintained by the receiver, which indicates that the cell is in the middle of a frame for that VP/VC pair, there is enough information to reassemble the frame.

The receiving card normally also needs a pointer to the location in host memory (or card memory if the card were to buffer received frames before DMAing them to the host, which it does not) where it has put previous cell payloads for incomplete frames, so that it can store future cells contiguously, or at least maintain a linked list. Once a SAR (segmentation and reassembly) engine design involves this leap from one bit to the size of a pointer, most go even further and use several more words for management purposes. VC table entries on the order of 16 to 32 bytes are not uncommon. Thus most ATM NICs are limited to on the order of 1024 VC/VP combinations active at a time.

Since OC3MON has no need for data beyond the first cell, and since it already maintains per-flow state on the host, we chose to limit the per-VC state on the card to the bare minimum: one bit (2 bits when we implement up-to-3-cell capture for OC12MON). This limit allows us to use 20 bits (19 bits for OC12MON) for VPI and VCI information, yielding a 128KB table size. Although the Fore cards have 256KB of memory, some of it is used for the i960 code (about 32K), the OS, reassembly engine data structures, and the stack. Since the VP/VC lookup needs an exact power of two, the largest we could get was 128KB. (Single-bit state for 2^{20} VP/VC combos = 2^{17} bytes = 131072 bytes = 128KB).

The cards for OC12MON will have 2MB of memory, all of which will be available $2^{24} = 16777215$ UNI (user to network interface) possible VP/VC combinations if we use one bit (or half that if we use two bits) of state per VP/VC. Note that when we copy multiple cells of the same packet to the host, the card will not place them near each other so the host must do further reassembly using the ATM headers.

Examining twenty bits of VCI/VPI information allows OC3MON to monitor over one million VCs simultaneously. The host controls exactly how many bits of the VCI this 20-bit index will include; the rest derive from the VPI. The host also specifies at startup what to expect for the remaining bits of the VPI/VCI, i.e., those not used for indexing into the card's state

table. The card can then complain about or at least drop non-conforming cells.

Many SAR engines choose to completely ignore the VPI and any bits of the VCI not used for indexing. When presented with the arbitrary VPI/VCI combinations we expect to see on a general purpose monitor, inevitable aliasing will cause collisions in reassembly state among VPI/VCI pairs. OC3MON avoids this situation by: (1) using a large number of VPI and VCI bits for its table lookup, leading to more successful reassemblies in the presence of arbitrary channel usage; and (2) comparing the bits it does not use for indexing with the expected values as described above, which keeps unsuccessful reassemblies from corrupting successful ones.

Since we want OC3MON to be able to see traffic on (almost) any VPI/VCI without prior knowledge of which circuits are active, and because the fast SRAM (static random access memory) used on such ATM cards for state tables is expensive and not amenable to modification by the consumer, this design turned out to be extremely advantageous.

Software: Description

The AAL reassembly logic is customized to capture and make available to the host only the first cell of each frame. The 48 bytes of payload from these cells typically contain the LLC/SNAP header (8 bytes), IP and TCP header (typically 20 bytes each). Copying the 5-byte ATM header also allows us the flexibility of doing ATM based analysis in the future. The SAR engine discards the rest of each AAL5 protocol data unit (PDU, equivalent to a frame or IP packet), limiting the amount of data transferred from the NICs over the PCI bus to the host. Although as yet unimplemented, one could increase the amount collected to accommodate IP options or larger packet headers as specified for IP version 6. Currently, however, the cards only pass the first cell of each packet, so when IP layer options push part of the TCP header into the second cell, these latter portions will not be seen by the host. Although suboptimal, we decided the savings in PCI (peripheral component interconnect) bus host memory and CPU usage justified this decision.

Each NIC (network interface card) has two 1MB buffers in host memory to hold IP header data. These cards are bus masters, able to DMA (direct memory access) header data from each AAL5 PDU into the host memory buffers with its own PCI bus transfer. This capability eliminates the need for host CPU intervention except when a buffer fills, at which point the NIC generates an interrupt to the host, signaling it to process that buffer up to memory while the NIC fills the other buffer with more header data. This design allows the host to have a long interrupt latency without risking loss of monitored data. The NICs add timestamps to the header data as they prepare to transfer it to host memory. Clock granularity is 40

nanoseconds, about 1/70 of the OC3 cell transmission time.

The resulting trace is conducive to various kinds of analysis. One could just collect a raw timestamped packet level trace in host memory, and then dump the trace to disk. This technique is useful for capturing a detailed view of traffic over a relatively brief interval for extensive future study. However, because we currently use the DOS-supplied disk I/O routines, which are blocking, we cannot write to disk simultaneously with performing flow analysis. In fact, the I/O is not even fast enough to sustain disk transfer of a packet trace without the flow analysis process running. Therefore one can only collect a trace as big as the size of host memory, which in our case would be 114MB (119.5 million bytes), and then must stop OC3MON header collection to let OC3MON transfer the memory buffer to disk. In the future we hope to develop separate I/O routines that directly use the hardware, bypassing the slower DOS routines, and allowing us to keep up with continuous collection and storage of full packet headers at OC3 line rate.

Because the amount of data captured in a packet level trace and the time needed for our disk I/O inhibits continuous operational header capture, the default mode of OC3MON operation is to maintain IP flow statistics that do not require the storage of each header. In this mode of operation, concurrently with the interrupt driven header capture, software runs on the host CPU to convert the packet headers to flows, which are analyzed and stored at regular intervals for remote querying via a web interface. The query engine we use is similar to that found in the NLANR FIX West (see <http://www.nlanr.net/NA/>) workload query interface. We will describe the methodology for deriving flow information in second half of the paper, followed by example snapshot statistics taken with OC3MON.

Internal OC3MON Data Transfer Rate

We tested OC3MON on an OC3c link fully occupied with single cell packets (as would occur in the admittedly unlikely event of continuous TCP ACKs with no data and LLC/SNAP disabled on the routers), which yields 353207.5 packets per second (or in the single-cell packet case, the same number of cells) across each half-duplex link. Each header, including timestamp, ATM, LLC/SNAP, IP and TCP headers consumes 60 bytes, so the internal bus bandwidth required would be $353207.5 \times 2 \times 60 \times 8 = 339\text{MB}$. The 32-bit, 33MHz bus in the PC is slated at 1.056 gigabits, so we do not expect bus bandwidth to be the bottleneck until we need to support OC12. There are already extensions to the PCI standard to double the bus width and speed, so when we need to support the worst case OC12 workload (i.e., single-celled traffic), the bus technology will likely be available. (Digital has already demonstrated the 64-bit part.)

Typical production Internet environments exhibit average packet sizes closer to 250 bytes (about five cells), and rarely full utilization in both directions of a link. If we estimate 66% utilization in one direction and full utilization in the other, we get a more realistic: $353207.5 \times 1.6666 / 5 = 117731$ headers per second, or 56.5 MB per second across the internal bus.

Sampling

We do not currently support sampling in the capture or flow analysis software. Although sampling is one option to avoid losing gaps of data during traffic burst, it changes the statistics of a timeout-based flow analysis in a most unclear way. For simply collecting packet headers, or to venture into the murky statistics zone that flows of sampled packet streams would involve, we could modify the card to support sampling in the future. Testing OC3MON with single-cell packets on OC3 indicated that we do not lose packets, so supporting sampling is not high priority for us at this time.

Security

Secure access is a problem for any machine, especially those capable of monitoring traffic. On both the vBNS and the commodity MCI Internet backbone, the monitoring machines live in locked machine rooms or secure terminal facilities. One can also require the monitor to only accept packets from known IP addresses, or configure the routers to block packets from unknown addresses from reaching it.

We obtain the flow data summary from OC3MON via a passwd-protected remote query to a port on the machine. This level of security is equivalent to that provided by most SNMP implementations. The query process triggers OC3MON to clear the current flow summary, but OC3MON retains the active flows in memory.

An Annex terminal server supports console one-time password access to OC3MON.

Part II: Methodology and Results

Flow Profiling Methodology

In deriving flow profile information from packets, we need to establish a definition of what constitutes a flow. Since the appropriate constraints to put on what one labels a *flow* depend on the analysis objective, our methodology specifies a set of parameters that are configurable based on the analysis requirements.

We specifically do not restrict ourselves to the TCP connection definition, i.e., SYN/FIN-based, of a flow. Instead, we define a flow based on traffic satisfying specified temporal and spatial locality conditions, as observed at an internal point of the network, e.g., where OC3MON sits. That is, a flow represents actual traffic activity from one or both of its transmission endpoints as perceived at a given network measurement point. A flow is active as long as observed

packets that meet the flow specification arrive separated in time by less than a specified timeout value, as Figure 2 illustrates. The lower half of the figure depicts multiple independent flows, of which many thousands may be active simultaneously at WAN transit points.

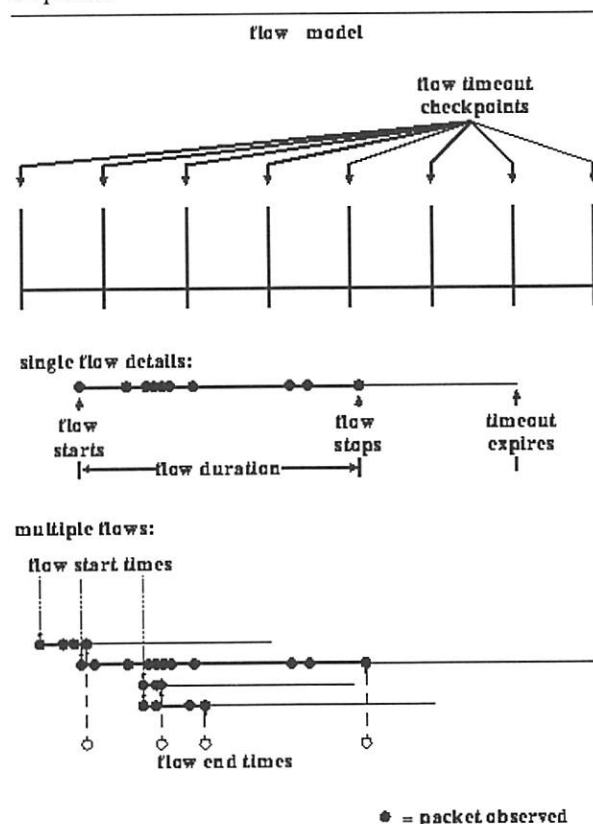


Figure 2: Defining a flow based on timeout during idle periods

This approach to flow characterization allows one to assess statistics relevant to issues such as route caching, resource reservation at multiple service levels, usage based accounting, and the integration of IP traffic over an ATM fabric.

Our definition of the timeout is similar to that used in other studies of timeout-based traffic behavior [4,6,8]. Jain and Routhier originally selected for their investigation of local network traffic a timeout of 500 milliseconds. Wide area traffic studies of the transport layer have typically used longer timeouts, between 15 and 20 minutes [8,10]. Caceres, et al. used a 20 minute timeout, motivated by the *ftp* idle timeout value of 15 minutes, and after comparison to a five minute timeout yielded minimal differences. Estrin and Mitzel [10] also compared timeouts of five and 15 minutes and found little difference in conversation duration at the two values, but chose to use a timeout of five minutes. Acharya and Bhalla [6] used a 15 minute timeout.

We explored a range of timeouts in Claffy, et al. [3], and found that 64 seconds was a reasonable

compromise between the size of the flow table and the amount of work setting up and tearing down flows between the same points. The timeout parameter is configurable in OC3MON, we have used the default of 64 seconds for the measurements in this paper. Initial tests with timeouts as large as 10 minutes did not significantly increase the number of flows, but we have not yet tested it under heavier data streams.

This timeout-based flow definition allows flexibility in how one further specifies a flow. There are other aspects that structure a flow specification: directionality, one sided vs. two sided, endpoint granularity, and functional layer.

Flow Directionality

One can define a flow as unidirectional or bidirectional. While connection-oriented TCP traffic is bidirectional, the profiles of the two directions are often quite asymmetric. Each TCP flow from A to B also generates a reverse flow from B to A, at the least for small acknowledgement packets. We define flows as unidirectional, i.e., bidirectional traffic between A and B would show up as two separate flows: traffic from A to B, and traffic from B to A.

One Versus Two Endpoint Aggregations of Traffic

This second aspect of a flow is related to the first. One can distinguish between single and double endpoint flows, that is, flows aggregated at the source or the destination of the traffic versus flows defined by both the source plus the destination. An example is the difference between all traffic to a given destination network number, versus all traffic from and to a specific pair of network numbers. Although single endpoint flows can be configured, OC3MON uses two endpoint flows by default, specifically at the host pair granularity.

Flow Endpoint Granularities

The third aspect of a flow is the endpoint granularity, or the extent of the communicating entities. Possible granularities include traffic by application, end user, host, IP network number, Autonomous System (AS), external interface of a backbone node, backbone node, backbone, or multibackbone environment (e.g., of different agencies or countries). These granularities do not necessarily have an inherent order, as a single user or application might straddle several hosts or even several network numbers. One example flow granularity of interest derives from the fact that IP routers make forwarding decisions based on routing tables that contain next hop information for a given destination network, a task implicitly grounded in one sided destination network layer flows at the granularity of IP network number. When policy routing issues render the source as well as the destination of a packet relevant to routing decisions, the issue of two-sided flow assessment is also important. Furthermore, as new routing mechanisms utilize alternative hierarchical definitions related to IP network numbers (e.g., CIDR masks), the desired granularity will likely shift.

Network administrators may want to define flows at a coarser granularity, such as aggregating network number pairs for which they create virtual circuits across their transit network. For example, an ATM cloud may bundle many finer grained IP flows within each ATM circuit. Conversely, a finer granularity would be necessary for providing special service to a single application instance, e.g., a videoconference.

These examples illustrate the importance of flexibility in the parameterization of a flow model, and the need to ground a flow specification in the requirements of the network, and even allow at any point in the network for multiple simultaneous flow specifications. One may want to assume flows: by destination network address for routing; by process pair for accounting; by source address for accounting and policy routing; by destination address or host or network address pair for bundling flows across ATM virtual circuits; or by address plus precedence information for flows at multiple priority levels.

Protocol Layer

Finally, there is the functional, or protocol, layer of the network flow. For example, one could define flows at the application layer. Alternatively one could

use transport connection information, e.g., SYN and FIN packets of the TCP protocol which support explicit connection setup and teardown. Because we want to maintain generality across all traffic, we consistently do not associate flows specifically with virtual connections, but rather define flows based on packet transmission activity based on specified endpoints at the network layer. Such a flow definition will not have a one-to-one mapping to active TCP connections; under certain conditions a single flow could include multiple active TCP connections, or a TCP connection may be contained in multiple observed flows over time. TCP traffic may furthermore be interleaved with UDP traffic, or a flow may consist entirely of non-TCP traffic.

Several factors motivate our decision to restrict ourselves to an observed state model, all reflective of one circumstance: the Internet is inherently a connectionless datagram environment, and thus connection oriented information cannot always be assumed available. We provide further details in an earlier study [3].

Configurability

These four aspects – directionality, one-sided vs. two-sided aggregation, endpoint granularity, and functional

MCI Flow Statistics Data Summaries

select below, then submit query

OPTION 1: ☒ Time-Series All Samples

OPTION 2: Specific Sample choose one:

838872001: 96/08/01 00:00:01	→
838872121: 96/08/01 00:02:01	→
838872240: 96/08/01 00:04:00	
838872360: 96/08/01 00:06:00	
838872480: 96/08/01 00:08:00	
838872600: 96/08/01 00:10:00	
838872720: 96/08/01 00:12:00	
838872841: 96/08/01 00:14:01	
838872960: 96/08/01 00:16:00	
838873080: 96/08/01 00:18:00	↙

If choosing a specific sample, also choose one of the following:

⌕ Suboption A: Printed Summary, sort by ☒ Flows ☒ Packets ⌕ Bytes ☒ Duration

☒ Suboption B: Graphic Profile of traffic types

☒ Suboption C: Country-Country matrix - Warning: this option is not yet operational (Sorry!) ☒ packets ⌕ bytes ☒ flows

☒ duration

(graph ⌕ or text ☒)

☒ Suboption D: Raw Data (ascii)

Figure 3: Main menu for OC3MON query engine

layer – provide a framework for specifying a flow profile structure. We designed the OC3MON flows analysis software to be flexible. One can specify a specific flow timeout (in seconds), an endpoint granularity (network, host, or host/port), and one- or two-sided flows (source, destination, or pair). We currently only support the classful IP network granularity, i.e., the most significant two bits of the address choose a netmask of 24, 16, or 8 bits. This is no longer appropriate in today's infrastructure, which instead uses CIDR (classless inter domain routing) variable length netmasks. Besides the network number, each routing update includes length of the netmask, which is unrelated to the contents of any bits of the IP address. OC3MON's next release will support the granularity of classless network by a CIDR-aware IP-to-AS-path mapping derived from a periodically updated dump of an actual routing table. This support will also include flow conversion to the autonomous system (AS) granularity, which will enable assessment of traffic flow at convenient macroscopic level.

One can also restrict OC3MON to analyzing flows for a specific transport protocol, port number, or host address.

For our flow profiling we use host pair plus source and destination application identifier (i.e., UDP/TCP port number), if they exist. That is, for the measurements in this paper OC3MON considers a flow unique based on its protocol, source IP address, destination IP address, source port, and destination port, and a 64 second timeout. A packet is considered to belong to the same flow if no more than 64 seconds have passed since the last packet with the same flow attributes. When flows time out, they are passed up to the statistics routines that update accumulators for remote querying via the Ethernet interface at regular intervals. The results of these queries, still in raw flow format, are then stored on a web server that supports a menu-driven interface. The menus, illustrated in Figures 3, 4, and 5, allow users to customize graphs of the data according to their interest.

Network Flow Traffic Statistics Time Series for MCI

Please Note – you want to make a choice in each of the 3 categories:

- Output Format
- Display (counts vs. proportion)
- What to Graph

Then submit the query

Output Format:

- ⬆️ `xmgr/xvgr graph` -- requires "data/xmgr; xmgr" in your mailcap file
- ✓ `tabulated spreadsheet` -- requires "data/spreadsheet" understanding by your browser
- ✓ GIF image, using 640 * 480 pixels

GIF image function not yet operational – sorry!

Display raw counts or proportions of total:

- ✓ display raw packet/byte/flow counts
- ⬆️ display proportion of total traffic

using data going back 10 days from 08 / 01 / 96

select below, then submit query

What to Graph:

- ✓ Packet Rate (overall pps)

Figure 4a: Time-series menu for OC3MON query engine (invoked upon clicking option 1 *time-series* in main menu)

Example Statistics

To illustrate the kind of graphs and tables one can retrieve, we provide sample graphs of OC3MON measurements on an OC3 trunk of MCI's IP backbone during the period between 29 July and 1 August 1996. We also provide international data from similar (earlier prototype) software running at the FIX West inter-agency interconnection point at NASA-Ames, in Moffett Field, California. Space restrictions prevent us from showing every graph type in this paper; we only provide a small set of possible plots to illustrate the utility of the tool.

Basic Counters: Packets, Bytes, Flows

Figures 3, 4, and 5 show the main menu and two submenu form interfaces for the query engine, respectively.

The main menu lets you choose either a snapshot of a specific monitoring interval (configured for two minutes here), or select one of the submenus. The snapshot returns a table similar to that shown in Table 1.

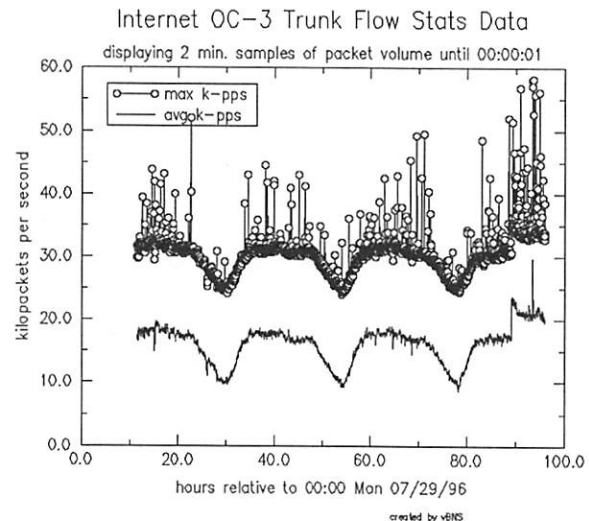


Figure 6: Average and maximum packets per second over two minute intervals on MCI IP OC3 backbone trunk Mon 29 Jul-Thur 1 Aug 96

What to Graph:

- ☒ Packet Rate (overall pps)
 - ☒ Bit Rate (overall kbps)
 - ☒ Packet Size (overall average packet size - will soon provide histograms)
 - ☒ Known Flows
 - ☒ Collection Loss Rate (collection losses)
 - ☒ IP Next Protocol
 - (☒ average packets and bytes per flow only)
 - ☒ Application (0=any) IP Next Protocol (e.g. 6 for TCP, 17 for UDP, 4 for IP-in-IP)

src port dst port

 - ☒ real audio (combined UDP ports 6970-7170 and TCP port 7070)
 - (☒ average packets and bytes per flow only)
 - ☒ Country-Country (sorry, this is not yet operational)
- 1: : 2: : ☒ packets ☒ bytes ☒ duration ☒ flows

☒ VRML object (bytes per flow per second over duration per flow)

This function is not yet operational

note: uses x-world/x-vrml for the object

-
-
-

Figure 4b: Rest of Figure 4a time-series menu

Figures 6, 7, 8, and 9 show packets, bytes, flows, and average per-second packet size for the four day

twenty thousand, with per second peaks as high as 58 thousand packets. The link utilization is around 50%,

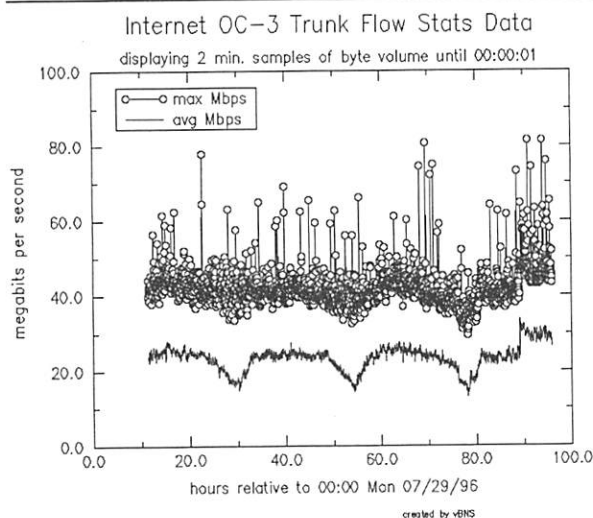


Figure 7: Average and maximum bits per second over two minute intervals on MCI IP OC3 backbone trunk Mon 29 Jul-Thur 1 Aug 96

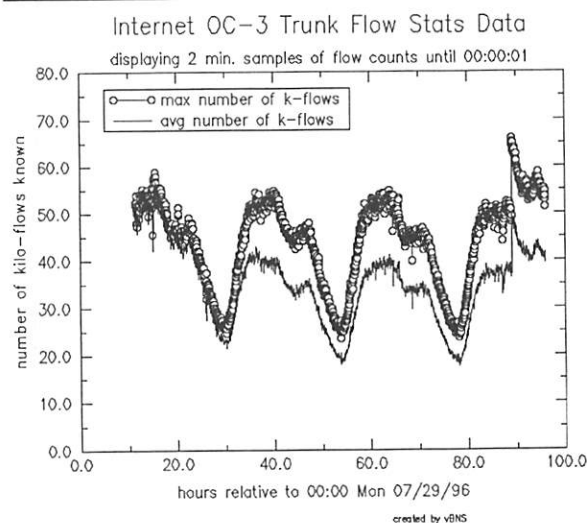


Figure 8: Average and maximum number of flows per second over two minute intervals on MCI IP OC3 backbone trunk Mon 29 Jul-Thur 1 Aug 96

period. During this measurement interval, the average number of packets per second cycled between ten and

which is a moderately loaded link in the current backbone. (MCI has also installed parallel OC3s where

graphing selected for measurement interval at BIPP-WTN1 00:00
Tue 08/13/96

select below, then submit query

output format for the result:

◆ gif image, using * pixels

◆ local graphing (requires xmgr or xvgr application and "data/xmgr; xmgr " or "data/xmgr; xvgr " in your mailcap file)

◆ packets over flows

◆ bytes over flows

◆ duration over flows

◆ packets over bytes

◆ packets per flow per second over duration per flow

◆ bytes per flow per second over duration per flow

◆ packets per flow over bytes per flow

Figure 5: 2D profiles menu for OC3MON query engine (invoked upon clicking suboption B *Graphic profile* option in main menu)

Flow Statistics Analysis for Internet OC3 Trunk
for date and time sample:08/14/96 at 01:20:01

Data Collection Summaries:

packet load (max/avg per second): 29461/15505
bit volume (max/avg per second): 41331776/8663743
packet sizes (max/avg per second): 244/194

flow summary:

maximum number of active flows: 44925
average number of active flows: 38718
average number of newly created flows per second: 535

error count/second (max/avg for three counters): 0/0 0/0 0/0
error percentage per second (max/avg): 0.00000/0.00000

Totals

trace duration: 299.502 seconds
total flow count: 160353
total packets: 3116705
total bytes: 861221626

Itemization by IP protocols

protocol	Flows	packets	bytes	duration	
	f-fract	p-fract	b-fract	d-fract	
1: ICMP	4755	36024	2753051	168691	absolute
	0.030	0.012	0.003	0.000	fraction
	76	7	578	35	ave pkt size,pkts,byts,dur
2: IGMP	3	1219	165730	886	absolute
	0.000	0.000	0.000	0.000	fraction
	135	406	55243	295	ave pkt size,pkts,byts,dur
4: IP	14	5192	913461	926	absolute
	0.000	0.002	0.001	0.000	fraction
	175	370	65247	66	ave pkt size,pkts,byts,dur
6: TCP	117449	2778274	808689615	-862387242	absolute
	0.732	0.891	0.939	0.500	fraction
	291	23	6885	-7342	ave pkt size,pkts,byts,dur
17: UDP	38085	295191	48604047	-864052262	absolute
	0.238	0.095	0.056	0.500	fraction
	164	7	1276	-22687	ave pkt size,pkts,byts,dur
47: GRE	10	500	51942	2727	absolute
	0.000	0.000	0.000	0.000	fraction
	103	50	5194	272	ave pkt size,pkts,byts,dur
83: VINES	5	70	4692	554	absolute
	0.000	0.000	0.000	0.000	fraction
	67	14	938	110	ave pkt size,pkts,byts,dur
93: AX.25	24	175	24386	776	absolute
	0.000	0.000	0.000	0.000	fraction
	139	7	1016	32	ave pkt size,pkts,byts,dur
148: unknown	7	59	14632	613	absolute
	0.000	0.000	0.000	0.000	fraction
	248	8	2090	87	ave pkt size,pkts,byts,dur
241: unknown	1	1	70	0	absolute
	0.000	0.000	0.000	0.000	fraction
	70	1	70	0	ave pkt size,pkts,byts,dur

Table 1a: Flow assessment snapshot of traffic during single interval of OC3MON collection (invoked upon clicking suboption A, *printed summary* in the main menu)

Application Details

Sorted by bytes

prot	source port	dst port	Flows f-fract	packets p-fract	bytes b-fract	duration d-fract	
=====	=====	=====	=====	=====	=====	=====	=====
TCP	80	0	45096 0.281	782256 0.251	362260508 0.421	717418 0.000	absolute fraction
				17	8033	15	per flow average
TCP	0	119	265 0.002	284514 0.091	178873180 0.208	33343 0.000	absolute fraction
				1073	674993	125	per flow average
TCP	0	80	50095 0.312	713105 0.229	40006387 0.046	613333 0.000	absolute fraction
				14	798	12	per flow average
TCP	20	0	516 0.003	54058 0.017	38851597 0.045	46553 0.000	absolute fraction
				104	75293	90	per flow average
TCP	0	25	3618 0.023	68213 0.022	22884422 0.027	-864616448 0.500	absolute fraction
				18	6325	203944101	per flow average
TCP	119	0	593 0.004	215040 0.069	19874375 0.023	39717 0.000	absolute fraction
				362	33514	66	per flow average
UDP	7648	7648	90 0.001	34543 0.011	14214500 0.017	11001 0.000	absolute fraction
				383	157938	122	per flow average
UDP	53	domain	31350 0.196	102557 0.033	13217739 0.015	-864419840 0.500	absolute fraction
				3	421	23536521	per flow average
TCP	5190	0	129 0.001	11878 0.004	6269981 0.007	20769 0.000	absolute fraction
				92	48604	161	per flow average
TCP	23	0	304 0.002	43325 0.014	5714030 0.007	28780 0.000	absolute fraction
				142	18796	94	per flow average
TCP	1091	0	1 0.000	3050 0.001	4531532 0.005	125 0.000	absolute fraction
				3050	4531532	125	per flow average
TCP	6667	0	685 0.004	16751 0.005	3384800 0.004	64961 0.000	absolute fraction
				24	4941	94	per flow average
TCP	25	0	2820 0.018	54382 0.017	3005756 0.003	54107 0.000	absolute fraction
				19	1065	19	per flow average
TCP	0	20	1052 0.007	65855 0.021	2850237 0.003	35584 0.000	absolute fraction
				62	2709	33	per flow average

Table 1b: Application details

traffic demands require them, and is currently installing the first OC12s.)

The average number of flows per second goes from around 20,000 at night to over 60,000 during the day. Note the average packet size goes in the opposite direction – the per-second average packet size gets larger at night, presumably due to less interactive traffic and likely occurrence of automatic backups.

Application Specific: Web, DNS, Mbone

OC3MON also supports analysis by TCP/UDP application type. Figure 10 illustrates the proportion of traffic from web servers using the well-known *http* port 80 (web servers can also use other ports, whose traffic will not be reflected in the graph) measured in packets, bytes, and flows. Note that web traffic consumes approximately the same proportion of flows as it does packets, but a somewhat larger proportion of

bytes, indicating the use of larger packet sizes relative to other Internet traffic.

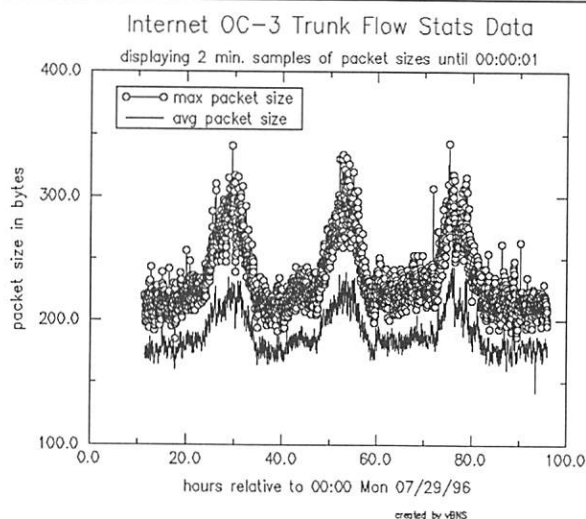


Figure 9: Average and maximum packet size over two minute intervals on MCI IP OC3 backbone trunk Mon 29 Jul-Thur 1 Aug 96

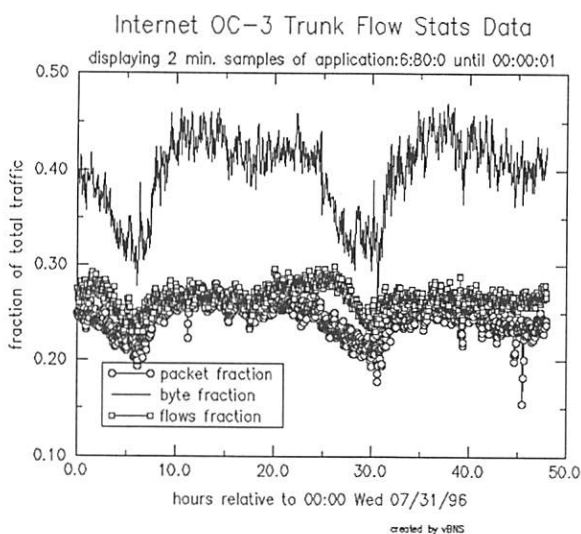


Figure 10: Proportion of web server-to-client traffic, i.e., from port 80 to any port, measured in packets, bytes, and flows over two minute sample intervals on MCI IP OC3 backbone trunk Wed 31 Jul-Thur 1 Aug 96

Figure 11 plots flows in the opposite direction, from clients to web servers; these flows have much lower byte proportions, being mostly query and acknowledgement traffic, slightly lower packet traffic, but similar flow counts.

Domain name system (*dns*) traffic is also characterized by short query/response packets and thus, as shown in Figure 12, comprises a huge proportion of (single packet, 40-80 byte) flows, but less than 8% of the byte traffic.

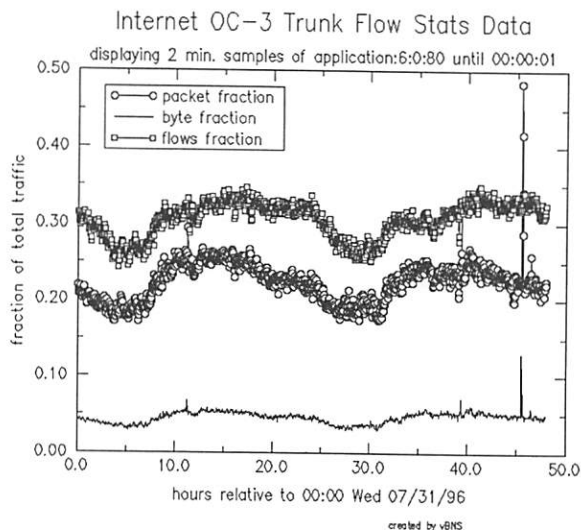


Figure 11: Proportion of web client-to-server traffic, i.e., to port 80 from any port, measured in packets, bytes, and flows over two minute sample intervals on MCI IP OC3 backbone trunk Wed 31 Jul-Thur 1 Aug 96

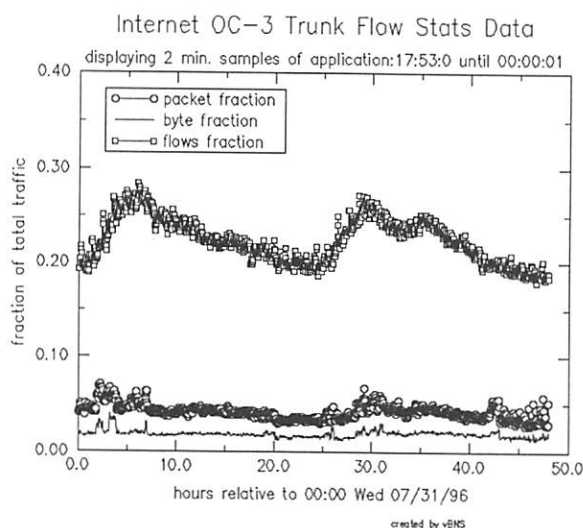


Figure 12: Proportion of *dns* traffic measured in packets, bytes, and flows over two minute sample intervals on MCI IP OC3 backbone trunk Wed 31 Jul-Thur 1 Aug 96

We can also look at the traffic by transport layer protocol; Figure 13 is the proportion of UDP packets, bytes, and flows (which includes all of the *dns* traffic plotted in Figure 12).

Figure 14 shows the absolute counts of IP traffic, again measured in packets, bytes, and flows. IP (IP protocol 4) traffic includes Mbone tunnel traffic, where very few flows each typically consume a substantial proportion of packets and bytes.

Although each mbone flow seems to consume an inordinate amount of resources, note that in the expected case, the mbone flows represent tunneled

multicast traffic, and thus potentially serve a larger number of customers than just the single flow depicts. In contrast, the *cuseeme* audio/video teleconferencing application, plotted in Figure 15 with a profile similar to the Mbone flow profile, is not multicast, and so poses a definite threat to Internet service providers trying to grow, or even maintain, a (still largely flat-priced) customer base.

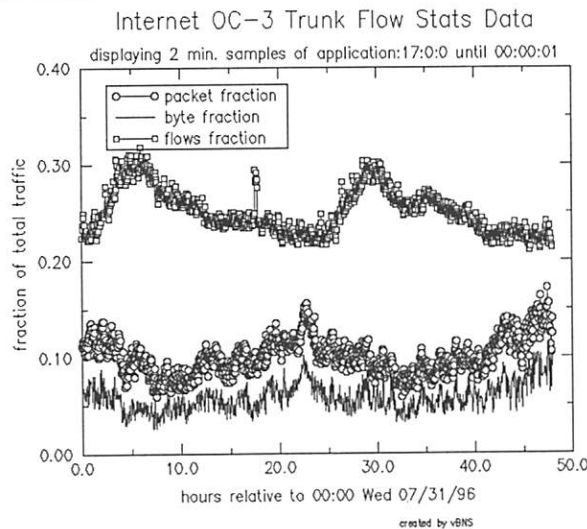


Figure 13: Proportion of *udp* traffic measured in packets, bytes, and flows over two-minute sample intervals on MCI IP OC3 backbone trunk Wed 31 Jul – Thur 1 Aug 96

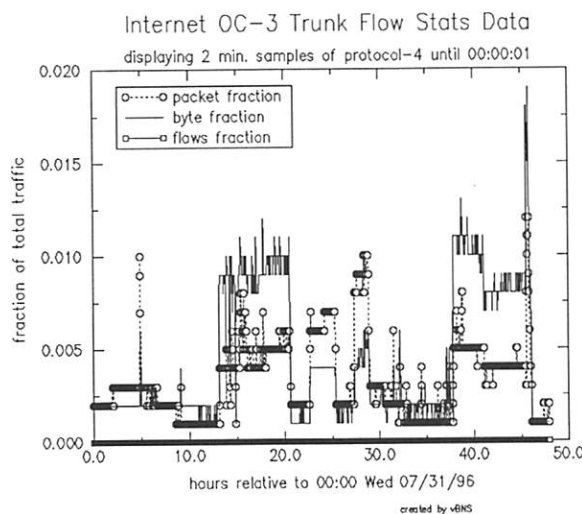


Figure 14: Proportion of *IP* (IP protocol 4) traffic measured in packets, bytes, and flows on MCI IP OC3 backbone trunk Wed 31 Jul-Thur 1 Aug 96

We might also want to know an average of how many packets and bytes are in a flow of a given type; Figure 16 shows this metric for *cuseeme* traffic.

The use of ports as an application classifier limits us to applications that use a single port. *Realaudio*

is an emerging application that uses more than one port: TCP port 7070, and UDP ports 6970 through 7170. Because we were particularly interested in the growth of this application, we modified the post-processing analysis script to support a query for this set of ports. (Note this will be an upper bound, since other applications may use those ports as well, e.g., AFS uses port 7000.)

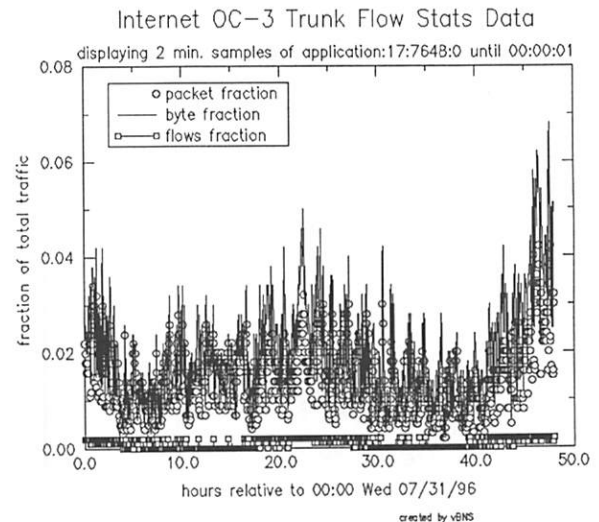


Figure 15: Proportion of *cuseeme* traffic measured in packets, bytes, and flows on MCI IP OC3 backbone trunk Wed 31 Jul-Thur 1 Aug 96

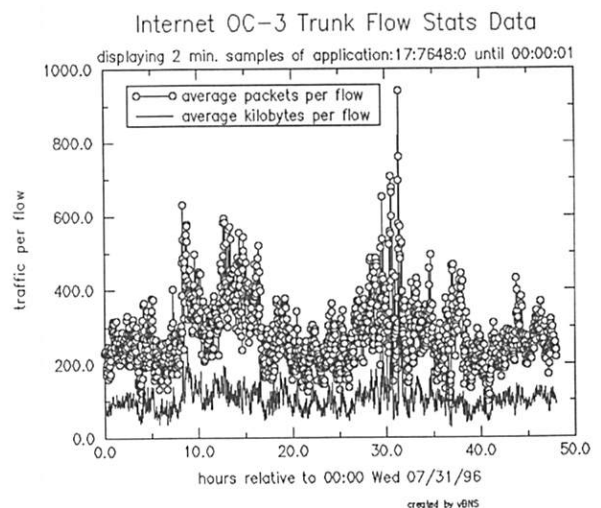


Figure 16: Average packet size of *cuseeme* traffic on MCI IP OC3 backbone trunk Wed 31 Jul-Thur 1 Aug 96

Profiles of Two Dimensions

The 2D profile menu shown earlier (Figure 5) allows one to classify application flows by two parameters at a time, e.g., their byte-duration or packet-duration products. This will allow figures such as that in Figure 18, which is an example from the FIX West location that has already implemented this feature.

Flow Across Geographic or Administrative Boundaries

OC3MON also is amenable to post-processing to derive geographic or administrative flow information. For example, we can look at the trade balance among countries. For the graphs in this section we use data collected from the FIX West interagency interconnection point at NASA-Ames, in Moffett Field, California. FIX West is a FDDI LAN medium that serves as a

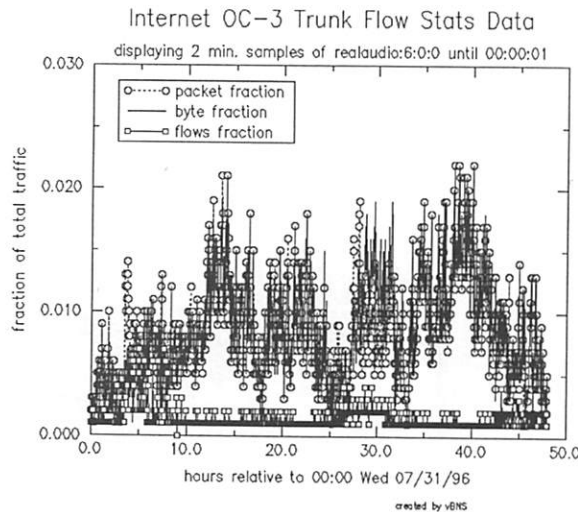


Figure 17: Proportion of *realaudio* traffic (which uses a set of ports: TCP port 7070, and UDP ports 6970 through 7170) measured in packets, bytes, and flows on MCI IP OC3 backbone trunk Wed 31 Jul-Thur 1 Aug 96

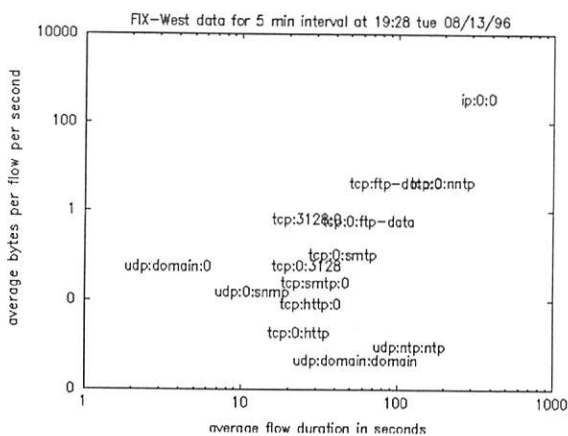


Figure 18: Byte-duration product of several popular applications during one 5-minute interval (FIX West)

network interexchange point among several providers, both national and regional, both commercial and federal. Although most commercial access has now moved to MAE-West, there are still several network providers that actively use the FIX: e.g., PACCOM, NSI, Sprint, MCI, ANS, and ESNET. All the following graphs reflect the second week of August 1996.

Clearly there is still a commodity left (IP packets) for which the United States is a net exporter to Japan.

Figure 19 shows IP traffic between the US and Japan throughout the second week of August 1996.

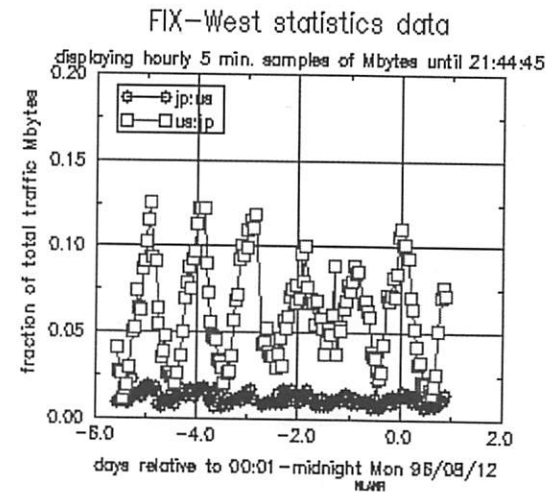


Figure 19: Japan-US trade balance in IP packets (FIX West, 7-12 August 1996)

Finally, one can graphically display traffic matrices by a specified granularity. We so far have only implemented support for country-by-country matrices, using the Internic database to map from IP address to country code. Figure 20 shows a snapshot of the top bandwidth-consuming countries from a single five minute sample from FIX West.

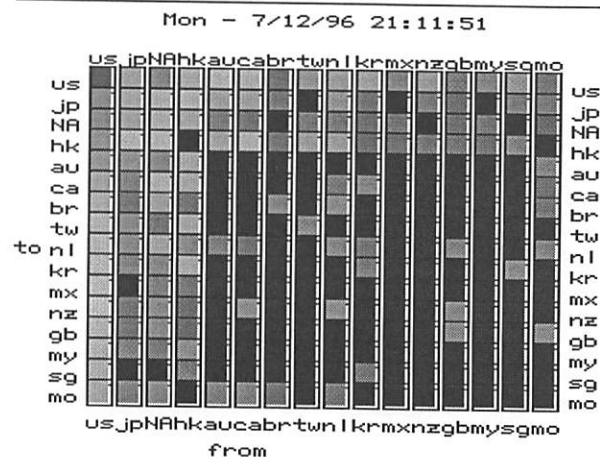


Figure 20: Country by country traffic matrix for FIX West five minute sample at 2100PDT 12 August 1996 (NA represents traffic of network numbers whose

country mappings were not available from the Internic database.)

Future Work

OC3MON's design is conducive to several extensions. Enhancements to the analysis of packet trace and flows data and the web interface to the data

are limited only by the imaginations of software developers. What we would most like to do next is to enhance OC3MON to use IP-to-AS path tables to support flow conversion at the autonomous system (AS) granularity, which will allow for assessment of traffic flow at a convenient macroscopic level.

Apart from the firmware on the NIC cards, OC3MON is not tied to the OC3 ATM interface. One can add any other interface type available for the PC, including FDDI and Ethernet. We are actively pursuing OC12c interface cards for the next generation of the monitoring platform. Our design goal is to be able to process both IP/ATM/SONET and IP/PPP/SONET encapsulations at OC12 rates with the same reasonably priced hardware.

We are also investigating moving more of the functionality of OC3MON, such as flows extraction from the packet header trace onto the interface card, in order to offload the host processor. This optimization may become increasingly useful at OC12c and OC48c speeds, where buses and host CPUs run out of steam. We believe that Field Programmable Gate Arrays (FPGA's) can provide this migration with a high degree of parallelism, without sacrificing the iterative design process and flexibility of software. We are also considering writing routines to access enhanced integrated drive electronics (EIDE) controller and the DMA engine on the Intel PCI ISA accelerator (PIIX3) directly to obtain much better asynchronous disk I/O.

Other extensions we examine with interest include:

- Real time graphic display of traffic behavior on a per application, per subnet, or per trunk basis.
- Re-creation of traffic patterns from a previously monitored packet trace
- WAN simulation by adding delay, jitter, and errors to a traffic stream

We hope to pursue collaborations with those interested in extending OC3MON's utility.

Conclusion

We have described the design, implementation, and use of a high performance yet affordable Internet monitoring tool. We have also described and shown examples from the web-based interface to the associated library of post-processing analysis utilities for characterizing network usage and workload trends. By using low cost, commodity hardware, we have ensured the practicality of using the monitor at a wide range of locations. Our network flow analysis tools have proven useful to us in understanding, verifying, debugging or spotting traffic behavior anomalies in the locations that we have deployed it.

Availability

The original prototype for web query engine, written by Hans-Werner Braun, is currently housed at <http://www.nlanr.net/NA/>. An electronic html version of this paper and pointers to the OC3MON software is at <http://www.nlanr.net/NA/Oc3mon/>. The software

itself is available via ftp from <ftp://nlanr.net/Software/oc3mon.zip>.

Acknowledgments

The data collection at FIX West is the result of a collaboration with NASA, MCI, NSF, and Digital Equipment Corporation. We are grateful to Hans-Werner Braun for prototyping the original flow statistics software, making it freely available, and coordinating its deployment at FIX West.

Author Information

Joel Apisdorf is a vBNS staff engineer at MCI. He dreamed up OC3MON in July of 1995 and has been implementing it ever since. He has developed communications and testing software for a variety of companies including Cable & Wireless, IBM, and GTE. Contact him via e-mail at apsidorf@mci.net.

kc claffy is a research scientist with the distributed National Laboratory for Applied Network Research, and based at the San Diego Supercomputer Center. Contact kc electronically at kc@nlanr.net.

Kevin Thompson is presently a senior engineer in the Internet Engineering department at MCI. He was employed as an engineer at MITRE Corporation in the Networking Center until 1995. He supports statistics collection architecture and implementation for the vBNS. His e-mail address is kthomp@mci.net.

Rick Wilder is an internet engineer at MCI. He manages MCI's engineering activities in the National Science Foundation's vBNS network and is involved in the planning and evolution of other IP services including internetMCI. His e-mail address is rwilder@mci.net.

References

- [1] k claffy and H. W. Braun, "Post-NSFNET statistics collection", *Proceedings Inet 95*.
- [2] k claffy and H W Braun, "Web traffic characterization: an assessment of the impact of caching documents from the NCSA's web server", *World Wide Web Conference*, 1994, Chicago, IL
- [3] k claffy, Hans-Werner Braun, George C. Polyzos, *A parameterizable methodology for Internet traffic flow profiling*, IEEE JSAC.
- [4] R. Jain and S. A. Routhier, "Packet trains - measurement and a new model for computer network traffic", *IEEE Journal on Selected Areas in Communications*, vol. 4, no. 6, pp. 986-995, September 1986.
- [5] J. Mogul, "Observing TCP dynamics in real networks", in *Proceedings of ACM SIGCOMM '92*, August 1992, pp. 305-317.
- [6] M. Acharya, R. Newman-Wolfe, H. Latchman, R. Chow, and B. Bhalla, "Real-time hierarchical traffic characterization of a campus area network", in *Proceedings of the Sixth International*

Conference on Modeling Techniques and Tools for Computer Performance Evaluation, 1992, University of Florida.

- [7] M. Acharya and B. Bhalla, "A flow model for computer network traffic using real-time measurements", in *Second International Conference on Telecommunications Systems, Modeling and Analysis*, March 24-27, 1994.
- [8] R. Caceres, P. Danzig, S. Jamin, and D. Mitzel, "Characteristics of wide-area TCP/IP conversations", in *Proceedings of ACM SIGCOMM '91*, September 1991, pp. 101-112.
- [9] C. Partridge, *A proposed flow specification*, Internet Request for Comments Series RFC 1363, September 1992.
- [10] D. Estrin and D. Mitzel, "An Assessment of state and lookup overhead in routers", *Proc. Infocom '92*.

IP Multiplexing by Transparent Port-Address Translator

Heon Y. Yeom, Jungsoo Ha, & Ilhwan Kim – Seoul National University

ABSTRACT

There are certain occasions when we want to have more than one computers using a single IP address. One notable case is when we use firewalls for security reasons or when we use private IP addresses and still want to provide global IP connection. Since the computers behind the firewall or using private IP addresses do not have unique IP addresses, they can not participate in global IP network. However, we still want to provide them some means so that they can be used as if they have real IP addresses.

Another example is when we have more than one server and want to make them appear to be just one server. With the proliferation of WWW, popular WWW servers tend to be overloaded and we need a mechanism to balance the load to several servers when needed. One solution being adopted is address translation by DNS (Domain Name Service). IP reuse using network address translator (NAT) is proposed in [TE93] and standardized by RFC1631. For load balancing of the WWW servers, a round robin distribution scheme using DNS is proposed and is used at NCSA [KBM94]. However, as pointed out from the literature, these schemes are based on DNS and there are several shortcomings. We propose a novel technique to perform address translation without a DNS. Our technique is based on the idea that any internet connection is uniquely identified by the socket pair <source IP address, source port, destination IP address, destination port>. By using a address translating bridge, we can provide computers with phony IP addresses real IP connection. The performance of our scheme is investigated using a prototype and the test results show that there is very little overhead in using our scheme.

Introduction

One of the biggest problems we are facing now on the Internet is the depletion of IP addresses. The four byte long IP address is being used up rapidly due to the exponential growth of Internet. We need to either expand the IP address space or devise an efficient way to use the existing IP address space.

Even though a new protocol called IPNG is being investigated to replace the IP, its wide acceptance is still far away and a short-term solution is badly required. We looked at some solutions suggested in the literature such as proxy, socks, and NAT. Among those solutions, NAT is considered to be the most practical since it does not need any changes in the network structure or current applications.

We propose a novel idea which we call port-address translator (PAT) which deals with the shortcomings of NAT that it still need IP addresses as many as the number of active local nodes at any time. By using PAT, only a single IP address is needed and it can support up to 2^{16} active connections.

The main idea behind PAT is that each TCP or UDP connection is uniquely identified by the internet address and port number pairs of the source and destination. Instead of having different IP addresses, one machine with a real IP address can act as a gateway by assigning different port number for each active connection.

To investigate the usability of the proposed PAT, we have conducted some experiments using our prototype. First, we investigated whether our PAT prototype can be used for all the protocols. In other words, the question is that the PAT is really transparent or not. We have found that PAT works well with most of the protocols except the protocols which use the endpoint information at the application level such as FTP, Rlogin and ICMP. In these protocols, the boundary between the communication layers is not clear and the lower layer information is used at the higher layer. To make PAT work with these protocols, we have to attack each protocols one by one and provide application specific services and we have solved the problem with FTP.

Secondly, we have examined the performance of our prototype and found that our prototype shows little overhead when used between two Ethernet segments.

As an additional benefit, PAT can be used as a load balancing gateway. Unlike the DNS based load balancing scheme which fails to balance the load from time to time due to DNS caching, PAT can distribute the loads to multiple servers perfectly at all times.

The rest of this paper is organized as follows: we identify the problem of IP address shortage and look at some of the solutions provided in the literature in section 2. In section 3, we propose our solution by translating IP address-port pair. Design and implementation issues are discussed in section 4 and we report the

results of our performance experiments in section 5. Finally, we conclude the paper in section 6.

Background

Waste of IP Address Space

Since there are four bytes in IP address, there can be 2^{32} distinct addresses. However, this address space is wasted due to the following reasons.

- To facilitate routing, IP addresses are grouped into network addresses.
- There are three different classes of networks classified according to their size (class A, B, and C) and each organization is assigned one or multiple contiguous network addresses.
- IP addresses inside a network address can not be used by other organizations.
- The number of nodes in any one network which simultaneously communicate with outside nodes is relatively small.

Think of a class C network which has 150 IP nodes in it. At a glance, at least 106 addresses are wasted because the unused addresses of a class C network cannot be used by the other sites belonging to another network. Moreover, if only 50 local sites tend to access remote sites simultaneously, we don't have to assign global addresses to each local sites all the time.

Thus, we insist that 206 out of 256 IP addresses are wasted in this example. Francis' NAT [TE93] can support this network with only 50 global addresses. In this paper, we present another application level gateway that can support this network with just one global address.

IP address Reuse

It has been already noted that IP addresses can be reused for private networks. In [RFC1597], parts of IP addresses are designated to be used repeatedly for private networks as follows:

- 1 class A network address 10.0.0.0
- 16 class B network addresses from 172.16.0.0 to 172.31.0.0
- 255 class C network addresses from 192.168.0.0 to 192.168.255.0

However, for nodes inside these private networks to work with Internet, there should be some mechanism to give them real IP addresses. Also, in [RFC1519], it has been suggested to use CIDR (Classless Inter-Domain Routing) to avoid IP address waste due to IP partitioning.

Proxy Server

For a local area network where network security is important, a firewall is often used to separate the inside network from outside networks. It is usually implemented by selectively filtering packets coming in and going out. However, the users inside the network still want to use all the Internet services, and it is a major consideration in using a firewall.

One easy solution is to provide a proxy server through which users communicate with outside network. By setting up a proxy server which can be trusted and assigning a real IP address to it, users can go through it when they need outside connection. Refer to Figure 1 for the operation of a proxy server.

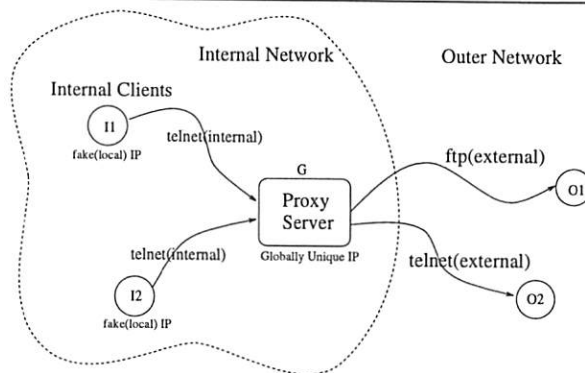


Figure 1: Connecting through a Proxy Server

Socket Server

In UNIX systems, application programs use sockets to access the network and each kernel of the system manages sockets. There are some exceptions where sockets are managed and allocated by a separate server to provide transparent network access where firewall is used. In [KK92], a socket server is introduced which manages the socket of the local machines which do not have direct internet access so that local machines can access outside network. The socket server should have the capability to communicate with outside network and it provides sockets which can be used for outside connection to the internal hosts which can not directly communicate with outside network. Internal nodes can get a socket from the socket server using Rconnect() call. The socket server provides the socket clients its IP port number and transfer packets through the port. Using the socket server and the socksified clients, it is possible to transfer packets between the internal and outside networks.

Even though this scheme provides transparent solution, it requires modification to all the client programs.

Network Address Translator

As we have seen earlier, it is impossible to connect from a regional network using replicated IP address to the global IP network without some tweaking. It is needed to translate the replicated address to a globally unique address to be able to connect to the global IP network since the regional addresses are only unique inside that region. We call these fake IP addresses as replicated addresses.

P. Francis has suggested a scheme for IP address reuse by transparent address translator using DNS in [TE93]. Figure 2 shows the network diagram of a regional network with replicated addresses and the usage of address translator.

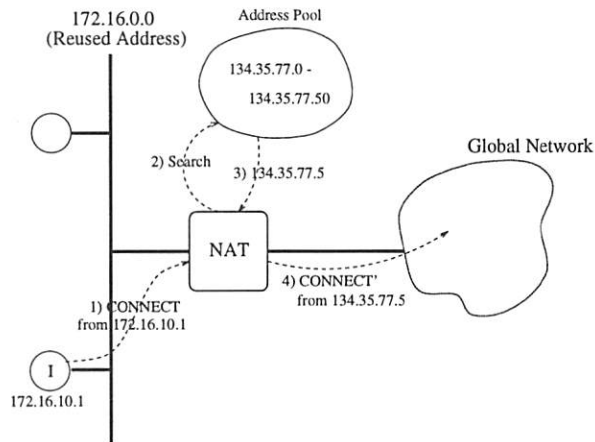


Figure 2: A network with replicated address and NAT

Between the global network and the local network using replicated IP addresses, there is a Network Address Translator (NAT) which has a couple of IP addresses which can be used for outside communication. Whenever a node inside the local network need a global address for outside communication, NAT provides the local node a global address dynamically. A global address, once allocated, can be used until the connection is terminated and the NAT automatically translates between the global address and local address. NAT performs the translation by intercepting all TCP/IP packets to search their header information and refer to the mapping table between the global and local addresses.

As shown in Figure 2, there is a close relationship between NAT, DNS and interdomain router or gateway. Let us look at how NAT works. When an inside node I wants to send a packet to outside network, NAT intercepts this packet and allocates a global IP address and relays it outside. All the packets to the outside network go through the NAT and the packets are examined and proper address translation is performed if needed. Only properly translated packets can be relayed to the outside network.

On the other hand, if there is a connection request from outside to the inside node I, DNS would first notice it and ask the NAT to prepare a global IP address for I. However, there is a set limit on the number of simultaneous outside connection which is the number of reserved global IP addresses the NAT has.

There are certain disadvantages in this scheme.

- It can be applied only to the applications using DNS.
- It is hard to apply it to connectionless protocol such as UDP.
- The number of global IP addresses might not be big enough for outside connections.
- A special consideration is needed for applications which exchanges IP address via the protocol. (FTP, RLOGIN, etc).

In spite of these drawbacks, this scheme enables a transparent address translation by setting up the NAT on the border of the network without a lot of modification. RFC1631 describes the use of NAT.

Transparent Port-Address Translator

Now, we introduce a new technique to re-use address space by transparent port-address translation. We note that there are a lot of ports in a node and only a small portion of them are used at any time. It is possible to provide outside connection by translating local address and port pair to a global IP address and its unused port number.

Let us denote a socket of a node as (address, TCP port number) and a TCP packet as (srcIP, srcPORT, dstIP, dstPORT). Assuming that there is no node with the same IP address on the route, every application using the connection can be uniquely identified in the regional network – in the global internet if there is no replicated addresses – by the pair (srcIP, srcPORT). Furthermore, two pairs of sockets, ((srcIP, srcPORT), (dstIP, dstPORT)) uniquely identify the two endpoints of the communication [RFC793].

Information Needed To Transmit Packets Through Proxy Server

Let us first find out the minimum information to transmit packets between TCP peer entities when proxy server is used. Let us look at Figure 3 which shows a telnet session through proxy server. From node S with replicated address, a remote login connection is established by issuing 'telnet G'. Again from G, 'telnet D' is used to connect to D and we have an indirect telnet connection from S to D. Proxy server G relays all the packets between S and D. As shown in Figure 3, a packet from (S, 1000) to (D, 23) is received by G and the source address is changed into (G, 3000) and then relayed to (D, 23). A packet from (D, 23) to (G, 3000) is again received by G and it is relayed to (S, 1000) after the destination address is changed. This translation is done by the proxy server when the user specified the usage of proxy server. Now, all we have to do is to transparently perform the translation done by the proxy server G.

Operation of Port-address Translator

We can automate the translation the proxy server does by using a special gateway. We call the gateway a Port-address translator (PAT). Figure 4 shows the operation of the PAT.

Since a port number is represented using 16 bits, an IP node can have up to 2^{16} different sockets. Therefore, there can be 2^{16} different entities on an IP node which can be uniquely identified in the Internet address space. In practice, only a small amount of ports are ever being used by applications. The remaining unused ports of a node with global IP address can be allocated to the regional nodes which have replicated addresses.

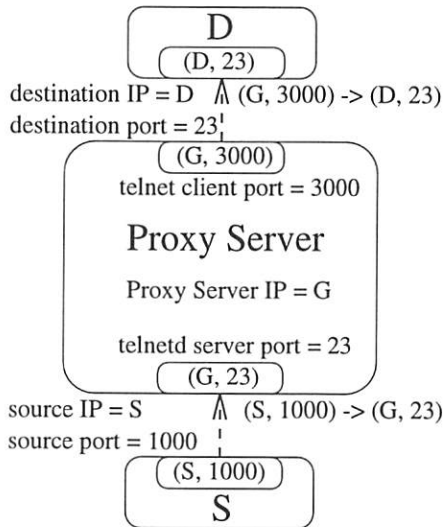


Figure 3: Packet relay by proxy server

G.PORT	I.IPAddress	I.PORT	STATUS
30000	172.16.10.1	1234	SYN-SENT
30001	172.16.10.1	3487	ESTABLISHED
30002	172.16.30.27	12039	CLOSE-WAIT
30003	NULL	NULL	CLOSED
30004	NULL	NULL	CLOSED
:	:	:	:
40000	172.16.68.99	3465	ESTABLISHED

Table 1: An example of port-address mapping table

Let us look at Table 1. This table shows the mapping between sockets (IPAddr, PORT) of nodes inside a regional network with stub B class network address 172.16.0.0 and the port numbers of a node G (Gateway node) which has a global IP address. This mapping can be dynamically allocated by:

1. DNS call as in [TE93]
2. Looking for SYN flag in TCP header information and deallocated when the STATUS becomes CLOSED.

Whenever PAT G encounters a packet requiring address translation, this table is looked up to modify the header information and the modified packet is relayed. G monitors all inbound packets and outbound packets to properly modify them.

The source address of the outbound packet is mapped from (I.Addr, I.PORT) to (G.Addr, G.PORT) according to the port-address mapping table and relayed. Also, inbound packets with destination address (G.Addr, G.PORT) is translated into (I.Addr, I.PORT) and relayed to the inside regional network. All the outbound packets should be translated before going out of the local network since the local IP addresses are replicated.

PAT as Load Balancing Gateway

With the growth of the internet, certain information servers tend to attract a lot of users and providing service to large number of internet users impose great strain on the popular servers. If the server is overloaded, the server should be upgraded or another server should be added to share the load. When using more than one servers, the question of how to distribute the load always comes up. The most widely used scheme is using DNS based load balancing. However, this scheme does not always work properly due to DNS caching. Using PAT, on the other hand, we can always distribute the load however we want. The load can be distributed round-robin fashion or the server load can be considered by simple programming of the PAT. Figure 5 shows the operation of PAT doing load balancing.

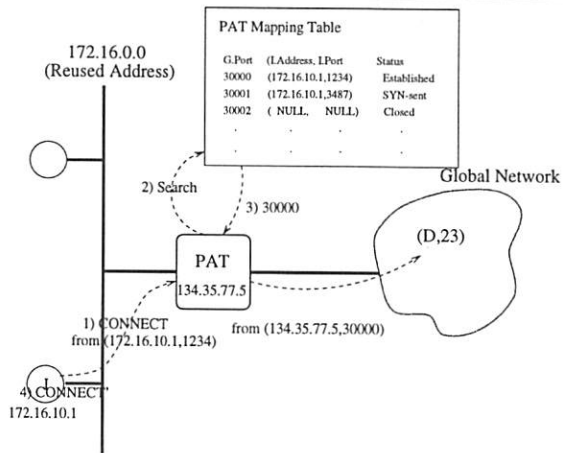


Figure 4: PAT operation as IP address multiplexer

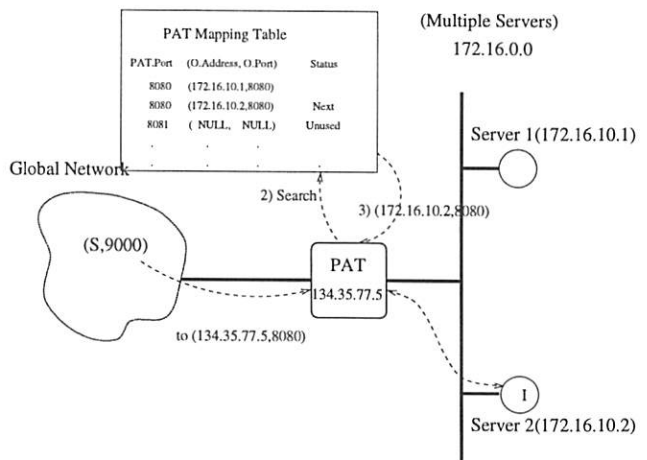


Figure 5: PAT operation as load balancing gateway

Design and Implementation Issues For Port-address Translator

We have implemented a prototype on an IBM compatible PC using packet driver. A lot of skeleton codes are adapted from PC bridge and PC router

softwares. Since the port-address translator has more functionality such as address translation, more hardware resource is needed. From the experiment we have done using the prototype, we found that the performance is comparable to that of general PC ethernet bridge.

Some of the issues we have encountered are as follows.

- The checksum of the TCP/IP header should be modified accordingly.
- It is difficult to decide when to allocate and deallocate mapping table entity for UDP protocol.
- A separate mechanism such as static allocation is needed for inbound request to the server with only regional address.
- To deal with application protocols which exchanges packets with IP address in them such as FTP and RLOGIN, the translation should be done on application layer.

These problems are due to the fact that the port-address translator can only gather limited information from the packets. They are similar to the problems encountered with packet filtering gateway which is used to enhance security. However, they are more difficult than the case with the translation using socket server since the information can be obtained from Rbind() call in socket server whereas the port-address translator only looks at the header information.

Calculating checksum for TCP/IP header is pretty simple and it can be efficiently done by a combination of additions and subtractions [RFC1631] [TE93]. We have implemented our prototype using the algorithm shown in [TE93].

Timing for Allocation and Deallocation of Port-address Mapping Table Entry

Two methods have been introduced for the timing of allocation and deallocation of port-address mapping table entry.

First, we can use DNS to dynamically allocate addresses. It has advantages that the address allocation time is specified and inbound service can be supported. However, it tends to be more complex since the address deallocation should be done either by monitoring each session or explicitly specifying the deallocation. Furthermore, for this method to work, all the nodes should be registered to the DNS and DNS caching can cause a severe inconsistency problem.

Second, we can monitor the packet header and its contents and modify them accordingly. It could be complex since we need to trace the regional sockets to follow their TCP STATE change. However, this information can be obtained from the TCP header flag (e.g., SYN, FIN). Figure 6 shows the state transition diagram of the TCP STATE and corresponding header flags. A pseudo code of the algorithm is attached as an appendix.

Supporting UDP

It is more difficult to extend the port-address translator to support UDP protocol. Since UDP is a stateless protocol and there is no sequence number, it is close to impossible to correctly trace a session. What we can do is to use an appropriate timeout value to determine a termination of a session. This idle time threshold based algorithm does not guarantees to work since the proper timeout value is hard to determine.

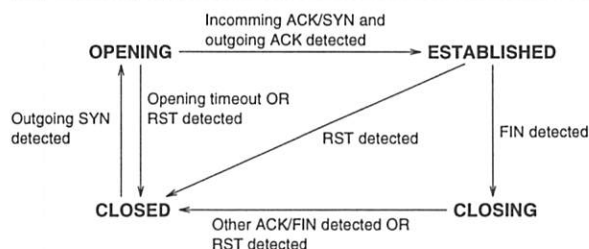


Figure 6: TCP STATE transition diagram from TCP header flag

We have implemented the idle time threshold based algorithm with a timeout value of two minutes. This value is selected since it is relatively large compared to the other timeout values used in NIS (Network Information System) or NFS (Network File System).

A special caution is needed to support UDP applications with different characteristics.

Another problem we have encountered with UDP protocol is the problem with large packets. When a UDP packet larger than 1472 bytes is transmitted through the Ethernet, it is divided into several packets before transmitted. The problem is that only the first packet has the UDP header. The remaining packets do not have the UDP header and the PAT has no way of knowing where to send these packets with no UDP headers. Since the first packet has the continuation flag, PAT can examine the flag whenever it receives a UDP packet and takes appropriate action when the packets with no header follow.

Supporting Inbound Request

Besides the dynamic allocation of the port-address mapping through packet flags, our port-address translator supports static allocations for inbound connection requests. It has a limitation that only one inside server can be specified for each IP address, port pair. In other words, only one server can be activated for each well known protocol if there is only one global IP address. The static allocation is assigned when the port-address translator is launched and stay valid all the time. We can have inside servers for well-known ports to service inbound requests.

For example, if we have a HTTP server at (www.our.domain, 80), packets coming to (gateway.our.domain, 80) is translated into (www.our.domain, 80) and relayed to the HTTP server inside. Of

course, our server should be registered as gateway.our.domain in the DNS.

Recently, information servers tend to be configured using a server pool to balance the server load. Our translator can be used for dynamic load balancing for a server pool as shown in the previous section.

Compatibility Issues

Since PAT changes the packet headers without the knowledge of senders or receivers of the packet, there are some network applications which do not work properly with the PAT. We have done extensive testing for various applications and found that most of the applications work well with the PAT. The TCP protocols that work with PAT are Telnet, SMTP, NNTP, HTTP, Finger, and X11. And UDP protocols that work with PAT are TFTP, DNS, and RPC. These protocols have common characteristic that they do not include IP address or port number in application layers. Even if they do, it is just for reference.

Some protocols that do not work with the current implementation of PAT are ICMP, NFS, rlogin and FTP. Most of these protocols use the IP address and port number in their application layers and the use of PAT renders the information useless. We have looked at each protocol to examine why they do not work and discuss some possible solutions.

ICMP

In ICMP protocol, the server is supposed to echo the header the client sends. Since PAT changes the address information in the header of the packet, the information inside the echo packet is different from the information that the client sent and the ICMP client does not recognize the echo from the server. One possible solution is to save a copy of the ICMP request header in PAT and switch the information inside the echo packet when the echo comes from the ICMP server.

NFS

NFS uses RPC and UDP protocol. Since both RPC and UDP work well with PAT, NFS should work with PAT. In our current implementation, UDP packets larger than 1472 bytes are not handled correctly and it renders NFS inoperable through PAT when accessing files of bigger than 1472 bytes. However, we believe NFS should work with PAT once the UDP packet handling is correctly implemented.

Rlogin

When we tried rlogin through PAT, we encountered an authentication failure. The rlogin server asks the client the address when the connection is established. Since the address of the client is different from the client's reply, it results in authentication failure. Again, we need to keep track of the initial negotiation and switch the client's address with the address of PAT to pass the authentication.

FTP

FTP is one of the most widely used protocols and it should be supported for the PAT to be of any use. There are two different types of connections used in FTP. The control connection for setting up the session and sending commands and the data connection for data transfer. The control connection is established when the FTP session is first set up while the data connection is separately established and terminated whenever there is a need for data transfer. When FTP client needs to get some data from the server, it sends a PORT command with its IP address and TCP port number for data connection through the control connection and waits listening to the port [RFC959]. Upon receiving the PORT command, the FTP server establishes data connection using the port number received.

When the client has a replicated address which is meaningless outside its regional network, it still sends a PORT command with its IP address and port number. For the client to be able to establish the data connection correctly, the port-address translator should recognize the packet carrying the PORT command and change the IP address and port number with a correct one before relaying this packet to the server.

This can be done by looking at the packets belonging to a FTP session and look for the PORT command with IP address and port number. Since all data are ASCII coded, it is relatively easy to find them. However, there is a slight problem since modifying the address and port number often results in different string length which causes inconsistent TCP sequence number. The inconsistency in TCP sequence number can render further communication impossible. The sequence number as well as the acknowledgement number of the ACK packet should be properly adjusted to ensure correct communication. In our prototype, we solved this problem by keeping track of the difference between the sequence numbers of the original packet and modified packet and properly adjust all the sequence numbers and acknowledge numbers for the forthcoming packets of the same FTP session.

Other Protocols

Applications like SNMP often uses encrypted IP address and port number. It is impossible to handle this kind of application which encrypts the packet contents at this stage. However, we believe that SNMP packets usually are confined inside its regional network and did not consider SNMP.

To be able to implement a transparent port-address translator, it is necessary to classify each applications and devise appropriate translation schemes for each of them. We are currently working on other applications we have overlooked using our prototype.

Performance Considerations

Test Environment

To measure the performance degradation the PAT might have, we have performed several experiments using our PAT prototype. The PAT prototype is implemented so that it will work between two ethernet segments and it is also possible to make it work between Ethernet and serial port so that it can act as a gateway machine.

First, we measured the Ethernet throughput on an isolated network between two Sparc 1 25MHz machines. Next, the PAT is introduced between two machines and the same set of experiments was performed. For the PAT prototype, we used an old IBM compatible PC with 386SX-25 and a 486DX-50 PC. The cost of the 486 PC used in the setup is about \$500 and the 386PC can be had less than \$100. It's always a good idea to use computers past their prime which have the highest price/performance ratio. The components needed for the PAT is as follows. (motherboard, CPU, RAM (640K if you can find or more), floppy drive, keyboard, power supply, LANcard X 2)

We have used two different workloads to run the experiments. The first is the TCP workload and the other is FTP workload to examine the effect of the FTP-specific service routine. We did not measure the effect of PAT on UDP workload since UDP is simpler than TCP and it would be less affected by PAT.

Workload T

To measure the TCP performance, we programmed a simple client-server application in which the client requests the server to send random amount of data. The connection is reset after the data transfer and established again until total of about 2MB data is transferred. The size of each request is randomly generated using exponential distribution of average 16KB and 160KB respectively. 16K was the average size of WWW objects observed by the proxy server in SNU. 160KB was used to see the effect of large data transfers.

Workload F

It is used to measure the effect of FTP-specific application. We measured the time to transfer files with total size of 7.2MB.

Two experiments for the workloads T and one experiment with workload F have been performed. For each experiment, we measure the throughput and response time of the transfer. We first measured the time without the PAT for the basis of comparison and measured the time with 486 PAT and 386 PAT respectively. To run the experiments, two sparc 1 workstations were used on an isolated network and the PAT was set up between them.

Test Results

Figures 7, 8, and 9 show the results of the experiments. When transferring large files using TCP, the

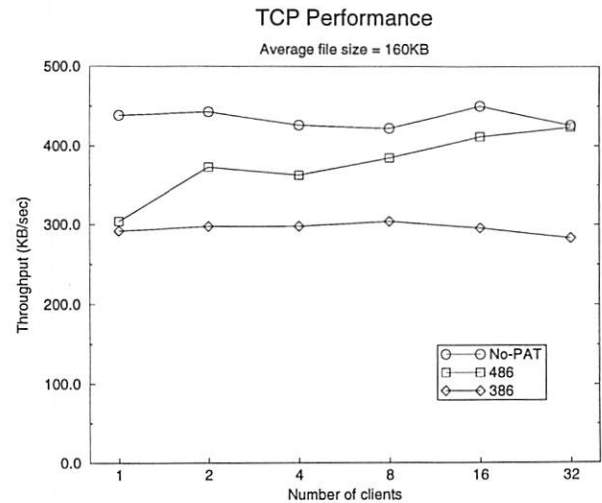


Figure 7: Performance of PAT (TCP, large file)

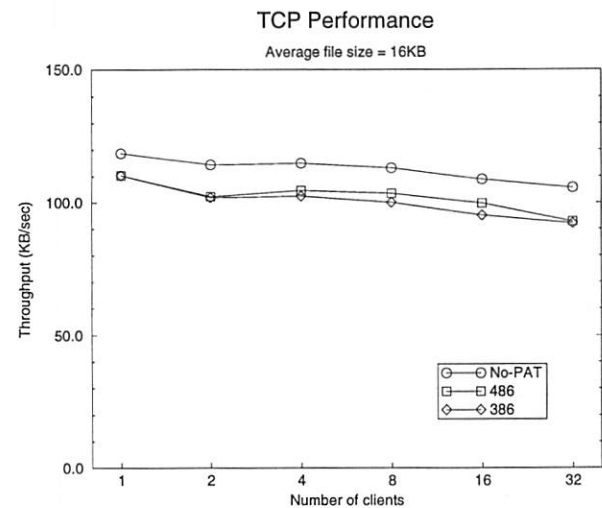


Figure 8: Performance of PAT (TCP, small file)

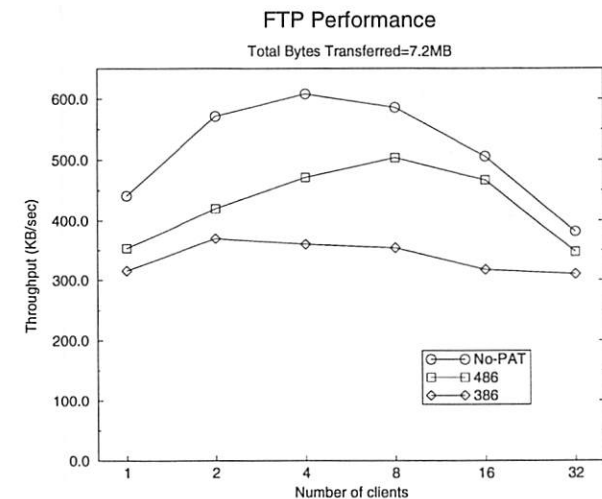


Figure 9: Performance of PAT supporting FTP protocol

386 PAT clearly shows poor performance since the slow gateway imposes great network delay. However, when using 486 PAT, the difference is relatively minor as the number of active connections increases. For small file transfers, there is less than 20% drop in throughput both for 386 and 486 PAT. We believe this is mainly due to the slow machines we used for the experiments. The machines rather than the network were the bottleneck in this experiment. As for the overhead dealing with FTP specific processing, we can see the same trend as with TCP large file transfer and we conclude that FTP processing does not impose big overhead in PAT operation.

We have also measured the round trip time of a small packet with and without PAT. Table 2 shows the result of that test.

In this experiment, we can see that 486 PAT only adds about 5% and 386 PAT has 10% more delay. Considering that it is round trip, for a packet to pass through PAT takes 0.00025 sec for 486 and 0.0005 sec for 386 PAT.

RTT between Sparc1	
no PAT	0.009287 sec
486 PAT	0.009784 sec
386 PAT	0.011348 sec

Table 2: Round trip time of a small packet (4 byte)

Overall, the 486 PAT only has less than 20% overhead with normal network traffic on Ethernet and we believe it would be more than enough to be used as a gateway for small sites connected through T-1 or slower links.

Conclusion

The port-address translator we propose can reuse IP addresses as well as prove load balancing without modifying existing applications or network structure. It is also very inexpensive. Compared to the address translator using DNS which provides one to one mapping between globally unique address and reused address, our scheme can provide a large number of connections using just one globally unique IP address. Ultimately, it can be used to transform a class C network to a network with only one globally unique IP address and reused addresses. We believe that the IP address reuse through port-address translator is a very practical solution and it can be a short term solution to the IP address shortage. The advantages of the port-address translator are summarized as follows:

- The cost of implementing a port-address translator and following network restructuring is comparable to that of packet filtering gateway.
- The port-address translator provide users transparent service without modifying existing applications programs or network topology.
- It is possible to relay UDP packets by careful considerations.

- It is possible to relay inbound requests by reserving ports or using server pool.
- Most of the applications such as FTP can be relayed by application specific gateway.
- It can provide security as a firewall does since it controls all outbound connections.

However, the port-address translator has the following drawbacks. These problems should be addressed before it can be a viable short-term solution for IP address depletion.

- There are some applications which need special consideration and some applications which can not be handled.}
- Various protocols such as ICMP, SNMP, RIP should be considered.}
- It has the single point of failure.

Availability

The prototype implementation of PAT is available from <ftp://arari.snu.ac.kr/pub/pat>. This version still does not handle long UDP packets correctly.

Acknowledgement

We thank Yongjae Lee for his sleepless nights implementing and debugging the prototype.

Author Information

Heon Y. Yeom received his B.S. degree in computer science from Seoul National University in Seoul, Korea and received his M.S and Ph.D. degree in computer science from Texas A&M University. While at TAMU, he also worked as a systems analyst in Texas Transportation Institute managing all kinds of computers. He currently is an assistant professor at Seoul National University and administers the computer center operations. Reach him electronically at yeom@arirang.snu.ac.kr.

Jungsoo Ha received his B.S. degree in computer science from Seoul National University and is presently working for his M.S. degree. He is one of the student systems administrators in SNU computer center and is responsible for the operation of SUN-SITE Korea and SNU proxy server. Reach him electronically at huggies@arirang.snu.ac.kr.

Ilhwan Kim received his B.S. and M.S degree in computer science from Seoul National University and is presently working for his Ph.D degree. He was one of the best student systems administrator in SNU and is now retired after he realized that it would be hard to earn his doctorate doing systems administration. Still, he intervenes from time to time as he sees fit. Reach him electronically at ilhwan@arirang.snu.ac.kr.

Bibliography

- [CB94] W. R. Ceswick and S. M. Bellovin. *Firewalls and Internet Security*, section 3, pages 49-83. Addison-Wesley, 1994.

- [KBM94] E. D. Katz, M. Butler and R. McGrath, "A Scalable HTTP Server: The NCSA Prototype", *The 1st International WWW Conference*, May 1994, Geneva, Switzerland
- [KK92] D. Koblas and M. R. Koblas. In *USENIX Security Symposium Proceedings III*, September 1992.
- [TE93] Paul F. Tsuchiya and Tony Eng. "Extending the IP Internet Through Address Reuse", *ACM Computer Communication Review*, January 1993.
- [Stev95] W. R. Stevens, *TCP/IP Illustrated*, Vol. 1, pp 148-151, Oct. 1995.
- [RFC792] J. Postel, *Internet Control Message Protocol*, 09/01/1981.
- [RFC793] J. Postel, *Transmission Control Protocol*, 09/01/1981.
- [RFC821] J. Postel, *Simple Mail Transfer Protocol*, 08/01/1982.
- [RFC854] J. Postel, J. Reynolds, *Telnet Protocol Specification*, 05/01/1983.
- [RFC959] J. Postel, J. Reynolds, *File Transfer Protocol*, 10/01/1985.
- [RFC977] B. Kantor, P. Lapsley, *Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News*, 02/01/1986.
- [RFC1034] P. Mockapetris, *Domain Names – Concepts and Facilities*, 11/01/1987.
- [RFC1035] P. Mockapetris, *Domain Names – Implementation and Specification*, 11/01/1987.
- [RFC1050] Sun Microsystems, Inc, *RPC: Remote Procedure Call Protocol Specification*, 04/01/1988.
- [RFC1258] B. Kantor, *BSD Rlogin*, 09/11/1991.
- [RFC1288] D. Zimmerman, *The Finger User Information Protocol*, 12/19/1991.
- [RFC1350] K. Sollins, *THE TFTP PROTOCOL (REVISION 2)*, 07/10/1992.
- [RFC1519] V. Fuller, T. Li, J. Yu, and K. Varadhan, *Classless Inter-domain Routing (CIDR): An Address Assignment and Aggregation Strategy*, September 1993.
- [RFC1579] S. Bellovin, *Firewall-Friendly FTP*, 02/18/1994.
- [RFC1597] Y. Rekhter and B. Moskowitz, *Address Allocation for Private Internets*, March 1994.
- [RFC1631] K. Egevang and P. Francis, *The IP Network Address Translator (NAT)*, May 1994.
- [RFC1813] B. Callaghan, B. Pawlowski, P. Staubach, *NFS Version 3 Protocol Specification*, 06/21/1995.
- [RFC1945] T. Berners-Lee, R. Fielding, H. Nielsen, *Hypertext Transfer Protocol – HTTP/1.0*, 05/17/1996.

Many Mail Domains, One Machine: The Forwarding Mailer

Hal Pomeranz – NetMarket/CUC International

ABSTRACT

With the explosion of domain registration in the last 18 months, many organizations find themselves having to maintain multiple separate email domains. Existing solutions for this problem are capital- or administrator-intensive. Here we explore a simple solution that allows an organization to support arbitrarily many separate domain spaces on a single host machine with only a single Sendmail server. This system has been used in production at the author's organization since 1994 with great success.

Background

Whether for fun or profit, many more organizations than ever before find themselves maintaining multiple domains. While BIND makes it very easy to configure and maintain multiple DNS domains, sendmail has historically assumed that every domain you have is simply an alias: if you have MX records for *foo.com* and *bar.com* pointed at the same machine, then *user@foo.com* is equivalent to *user@bar.com*. For many organizations (ISPs, Web development shops, and private citizens providing Internet access for friends, to name a few) this assumption is no longer sufficient.

So, your organization has registered *foo.com* and *bar.com* and you want it to be the case that *user@foo.com* to be somebody different from *user@bar.com* (at least from an email perspective). Historically there have been three different types of solution for this problem:

1. Run multiple physical servers. The easiest solution from a sendmail configuration perspective is to simply run a different physical piece of hardware for each distinct mail domain you require. From an administrative perspective, however, this model simply doesn't scale to dozens of domains. This approach also doesn't make sense from a capital cost and resource utilization perspective-- 90% of the time your dozen mail servers are mostly idle, while perhaps one or two of them are overloaded.
2. Run multiple copies of sendmail on one machine. Multiple copies of sendmail bound to different ports with a single master process to handle dispatching mail appropriately can simulate the effect of having multiple distinct hardware components. However, not only does this approach require guru-level knowledge of sendmail configuration, running dozens of sendmail processes simultaneously is incredibly memory inefficient.
3. Build a multi-zone class DB. Under sendmail v8, it is possible to build a class database which

causes a single sendmail daemon to route mail for *user@foo.com* to be delivered to a different place than mail for *user@bar.com*. You must, however, exhaustively and explicitly list all addresses which require special handling: you may not apply rules which apply to all *user@host* addresses within a particular domain. Imagine the impossibility of maintaining this scheme if you are an ISP providing mail services for dozens of clients, or a mail administrator for a nation- or world-wide company.

Now assume that, like everybody else in the world, you are an organization with limited capital budget, limited computing resources, and without the administrative resources to maintain a very complex system on an ongoing basis. None of the choices above are very appealing.

Requirements

An ideal system to meet this set of requirements should:

- Never lose mail or generate bounce messages to sender
- Not require a different machine for each domain
- Not require operating a separate sendmail process for each domain
- Simultaneously allow both specific (*user@host.foo.com*) and global (*all user@host pairs in foo.com*) forwarding rules
- Allow addition of new forwarding rules without modifying *sendmail.cf* or restarting active sendmail processes
- Be as simple to administer as */etc/aliases*
- Not require modification to the sendmail source distribution
- Not significantly increase delivery time of mail or put undue load on mail server

Our solution was to add an entirely new mailer to our *sendmail.cf* file (though this mailer uses the

same sender/recipient rules as the *prog* and *local* mailers). "Delivery" via the forwarding mailer is triggered by the presence of the domain in a new class file (*/etc/mail/forward.for* on Solaris systems). The forwarding mailer is actually a Perl program which uses an external file of regular expression based rules to determine the actual delivery address for *user@host.foo.com*. Once the real delivery address is determined, the forwarding mailer simply hands the message back to sendmail for delivery.

Implementation

While one never approaches a project like this without some large amount of trepidation (or perhaps "fear and loathing" is a more apt description), the standard m4 tree provided with sendmail v8 makes the whole configuration process a breeze.

The first step is to add a new mailer definition (sendmail expects these definitions to reside in its *cf/mailer* directory). Figure 1 shows the complete *cf/mailer/forw.m4* definition file. Notice that the file contains more comments than actual configuration lines. Several parameters are defined at the top

of this file, but they should not be customized here (or modified at all, in most cases). The *ifdef* syntax used in *forw.m4* means that site administrators can change these macros in their site configuration files and override the default values given here.

The second step to configuring your mail host to use the forwarding mailer is to modify your machine-specific m4 configuration file. Add the lines from Figure 2.

The first line demonstrates how the site administrator would change the values defined at the top of *cf/mailer/forw.m4* (in this case to locate the forwarder program to a hypothetical read-only device for security). The *MAILER(forw)* directive causes the configuration information that is stored in *cf/mailer/forw.m4* to be included in *sendmail.cf*. Next we define a sendmail class F from the file */etc/mail/forward.for* (more on this file in a moment).

The last two lines add new rules in ruleset 0 which cause the forwarding mailer to be invoked if the recipient address matches one of the domains in */etc/mail/forward.for* (don't forget that you

```
ifdef('FORW_MAILER_FLAGS',, 'define('FORW_MAILER_FLAGS', 'DFhMnsu'))
ifdef('FORW_MAILER_PATH',, 'define('FORW_MAILER_PATH', /etc/mail/forwarder)')
ifdef('FORW_MAILER_ARGS',, 'define('FORW_MAILER_ARGS', 'forwarder $u@$h'))

#####
### NetMarket Forwarding Mailer Hack ###
### Copyright (C) NetMarket/CUC International, 1995-6 ###
### All rights reserved. ###
### Permission to freely distribute under the same terms as Sendmail. ###
### No warranty expressed or implied. ###
#####

Mforw,          P=FORW_MAILER_PATH,
                 F=FORW_MAILER_FLAGS, S=10, R=20/40, D=$z:/tmp,
                 A=FORW_MAILER_ARGS

#####
# Uses Sender and Recipient rules from prog/local mailer definition
#####
```

Figure 1: Complete *forw.m4* definition file

```
define('FORW_MAILER_PATH', '/read-only/bin/forwarder')
MAILER(forw)

LOCAL_CONFIG
# Used by NetMarket forwarding mailer hack
FF/etc/mail/forward.for

LOCAL_RULE_0
# pass things off to forwarding mailer
R$* < @ $=F > $*          $#forw $@ $2 $: $1
R$* < @ $=F . > $*         $#forw $@ $2 $: $1
```

Figure 2: Machine specific m4 configuration file

MUST HAVE TABS between the left and righthand columns in these rules). Note that we are requiring an explicit match: if you only have `foo.com` in the `forward.for` file, the forwarding mailer will NOT be invoked on `user@host.foo.com`. Because large numbers of deeply nested subdomains are unusual (due mostly to the inconvenience of having to type and remember `user@host.location.org-unit.domain.com`), we have found that this configuration decision has made administration simpler without significantly increasing the size of the `forward.for` file.

Any additional site-specific configuration parameters should be added to the above rules, and a `send-mail.cf` file generated using `m4`. Install this file in the appropriate location for your system (`/etc/sendmail.cf` and `/etc/mail/send-mail.cf` are common).

The third step in the installation process is to use `/etc/mail/forward.for` to list all of the domains you want the forwarding mailer to be invoked on. Here is a sample `forward.for` file

```
beeper.netmarket.com
infersys.com
ftd.com
world.ftd.com
```

One domain or host address per line, any number of lines. Comments and blank lines are not allowed. Note that you kill and restart `sendmail` every time you update this class file, which (along with being unable to enter comments in the file) we consider to be a defect in `sendmail v8`.

Step number four is to install the forwarding mailer program as `/etc/mail/forwarder` (or whatever the definition of `FORW_MAILER_PATH`). The complete code for the script is given as Appendix A. At the top of this script are three variables

```
$patfile = '/etc/mail/patterns';
$error_recip = 'postmaster';
$mail = '/usr/lib/sendmail';
```

The first line tells the forwarding mailer where to look for its forwarding rules. The second line gives the recipient for error messages so that the sender does not receive bounces from the forwarding mailer (errors include being unable to open the patterns file or to find a match for the recipient address, as well as the recipient address containing shell or Perl meta-characters). The last line gives the explicit path to `sendmail`. No additional configuration of the forwarder executable is required.

Forwarding Rules

The fifth and final step of the installation is to create your own `/etc/mail/patterns` file. The file is in a two-column format, with a Perl regular expression on the lefthand side and one or more recipient addresses on the right hand side. The forwarding mailer scans this file in order (ignoring blank lines and lines starting with `#`) until it finds a regular expression which matches the recipient address of the original message, and the mail is then forwarded to the recipient list specified on that line of the patterns file.

Note that this “first match and exit” behavior is completely unlike the way `sendmail` processes `sendmail.cf`. Also, the columns in the patterns file are whitespace separated, as opposed to being rigidly tab separated as in `sendmail.cf`. The downside of this flexibility is that the regular expression on the lefthand side of the patterns must not contain any whitespace (use the Perl `\s` operator if you really need to match whitespace).

One use of the forwarding mailer is to define pseudo-subdomains within your own organization. For

```
####
#### beeper.netmarket.com -- people to annoy
####

(?i)^hal@beeper.netmarket.com$      beep-hal@netmarket.com
(?i)^jxh@beeper.netmarket.com$      beep-jxh@netmarket.com
(?i)^jim@beeper.netmarket.com$      beep-jxh@netmarket.com
(?i)^josh@beeper.netmarket.com$     beep-josh@netmarket.com
(?i)^kevin@beeper.netmarket.com$    beep-kevin@netmarket.com
(?i)^sysadmin@beeper.netmarket.com$ beep-sysadmin@netmarket.com
```

Figure 3: *beeper.netmarket.com* pseudo-domain

```
####
#### beeper.netmarket.com -- people to annoy
####

(?i)^jim@beeper.netmarket.com$      beep-jxh@netmarket.com
(?i)^@beeper.netmarket.com$         beep-$u@netmarket.com
```

Figure 4: Alternate configuration

example, we have a *beeper.netmarket.com* domain which enables us to send messages to a user's alphanumeric pager by sending mail to *user@beeper.netmarket.com*. Figure 3 shows the appropriate lines from the *patterns* file. Of course, there's no real value to this pseudo-domain other than we found it easier for people to remember *user@beeper* than *beep-user*.

The forwarding mailer allows the recipient list on the righthand side of the *patterns* file to contain *\$u* tags which get expanded to be the username portion of the original message recipient. Using *\$u* we can rewrite the six lines above as Figure 4. Note that we are not only utilizing the *\$u* tag but also the "first match" property of the *patterns* file.

Handling forwarding for full toplevel domains with many users, mailing lists, and auto-responders can get tricky. Due to security concerns, the forwarding mailer does not handle piping mail to Majordomo or other autoresponder-type programs. We've adopted the convention of Figure 5 for handling these addresses. The */etc/mail/aliases* entries for *<thing>-at-infersys@netmarket.com* not only feed the mail to Majordomo or our locally developed autoresponder as appropriate, but also archive the original message in a file.

Note that we don't want to use *\$u* to simply forward every *infersys.com* address to *\$u-at-infersys@netmarket.com* because we also have entries like those in Figure 6. The first line forwards mail for a specific user (in this case, the owner of the domain) to a user in the *netmarket.com* domain. The second line

(the default) catches all other mail to users in *infersys.com* and sends it to the domain owner (triggering a second trip through the forwarding mailer when send-mail attempts to resolve *irilyth@infersys.com*).

Some of our forwarding domains are merely aliases on top of some other ISP's domain space as seen in Figure 7. These lines allow us to intercept mail for *webmaster* and *postmaster* for the given domain but then pass through all other mail to user accounts at some other ISP. We added the last line because a lot of people seemed to get confused between *world.ftd.com* (a non-existent host) and *world.std.com* (a large public-access Unix site in the Boston area); we created an auto-responder to send them a polite message.

The last lines of our *patterns* file are used to catch addresses that haven't matched any explicit patterns; see Figure 8. The first three lines grab all mail for *webmaster*, *postmaster*, *hostmaster*, *root*, or *MAILER-DAEMON* and deliver them locally. The last line forwards all other mail to an autoresponder which sends out a polite "nobody home" response.

Appendix B gives the complete text of a sample *patterns* file using the above examples.

Our Experiences Using the Forwarding Mailer

We have been operating the forwarding mailer for over a year at this point, supporting dozens of domains and subdomains with thousands of users at our high-water mark. The system has been incredibly robust and exceeded all of our requirements. The

(?i)^majordomo@infersys.com\$	majordomo-at-infersys@netmarket.com
(?i)^majordomo-owner@infersys.com\$	majordomo-owner-at-infersys@netmarket.com
(?i)^boston-b5@infersys.com\$	boston-b5-at-infersys@netmarket.com
(?i)^owner-boston-b5@infersys.com\$	owner-boston-b5-at-infersys@netmarket.com
(?i)^info@infersys.com\$	info-at-infersys@netmarket.com

Figure 5: Convention for handling piping mail

(?i)^irilyth@infersys.com\$	irilyth@netmarket.com
(?i)^.*@infersys.com\$	irilyth@infersys.com

Figure 6: Trap other addresses

(?i)^webmaster@ftd.com\$	webmaster@netmarket.com
(?i)^postmaster@ftd.com\$	postmaster@netmarket.com
(?i)@ftd.com\$	\$u@some-isp.net
(?i)@world.ftd.com\$	ftd-bounce-bot@netmarket.com

Figure 7: Aliases for forwarding domains

(?i)^(web host post)master@.*\$	\$u
(?i)^root@.*\$	root
(?i)^mailer-daemon@.*\$	mailer-daemon
(?i)^.*\$	bounce-bot@netmarket.com

Figure 8: Catch-all patterns

forwarding mailer was so successful, in fact, that it was regarded as a core technology that gave us significant "competitive advantage" over other Internet development firms, and the author was enjoined from releasing it at the last LISA Conference.

Currently, our mail server (a Sparc 20 with 256MB of RAM) handles 200-250 distinct email messages per hour during working hours, approximately 10% of which move through the forwarding mailer. Overall, we process 4000-5000 email messages per day. This (relatively low) mail volume has no significant performance impact on the mail server, which is also used as a general purpose development machine.

A trip through the forwarding mailer adds less than one second to the overall delivery time of mail that ends up being delivered locally. Obviously, the cost of passing through the forwarding mailer is higher if the mail comes from an external address and must be sent back out through our firewall for delivery at an external site.

Adding new rules to the `patterns` file is straightforward: even administrators without a background in Perl regular expressions can easily cut-n-paste and adapt rules from elsewhere in the file. Initial setup of the system could be cleaner in some respects, but can be forgotten after the initial configuration is done. Having to kill and restart `sendmail` when updating the `forward.for` class file is particularly annoying.

Availability and Future Plans

A sharfile containing the software (with limited documentation) is available at <ftp://ftp.netmarket.com/forw-mailer/forw-mailer.shar>. This software is provided "as-is" and is freely distributable under the same terms as `sendmail`, provided all copyright statements are retained.

In the future, there are several avenues for improvement of the software:

- Too much configuration is required in the `m4` file. The `LOCAL_CONFIG` and `LOCAL_RULE_0` lines should merge into the mailer definition.
- The `forwarder` program could be rewritten in a compiled language to improve execution speed and reduce performance impact.
- The documentation can be improved significantly.

The author is particularly interested in any sites that employ the forwarding mailer on high-volume mail servers (more than 100,000 messages processed per day).

Author Information

Hal Pomeranz (hal@netmarket.com) is Director of Infrastructure Development/Support and Information Security Officer at NetMarket, the Internet

division of CUC International. He has been active in the system and network management/security field for over ten years, and is a regular speaker and organizer for technical conferences and professional gatherings. Hal is a columnist for *USENIX* and other technical publications and has served on the Board of Directors for both the BayLISA and BBLISA System Administration professional societies.

Appendix A: forwarder Source Code

```
#!/usr/local/bin/perl
$patfile = '/etc/mail/patterns';
$error_recip = 'postmaster';
$mail = '/usr/lib/sendmail';

#####
#### NetMarket Forwarding Mailer Hack #####
#### Copyright (C) NetMarket/CUC International, 1995-6 #####
#### All rights reserved. #####
#### Permission to freely distribute under the same terms as Sendmail. #####
#### No warranty expressed or implied. #####
#####

use Syslog;

$error_str = '';

$toaddr = $ARGV[0];
$error_str = "$0 called with null argument" unless ($toaddr);
$error_str="Hostile address $toaddr" if ($toaddr =~ m, [\{\}\(\)\ \/\; \& \|\$ \=] ,.);
($touser, $tohost) = split(/@/, $toaddr);

unless ($error_str) {
    if (open(PAT, $patfile)) {
        $match = 0;
        while (<PAT>) {
            next if (/^#/ || /\s*$/);
            ($pattern, $recip) = /^(\\S+)\s+(.*\\S)\\s*$/;
            next unless ($pattern && $recip);
            if ($toaddr =~ /$pattern/) {
                $match = 1;
                last;
            }
        }
        close(PAT);
        if ($match) {
            $recip =~ s/\\$u/$touser/g;
        }
        else { $error_str = "No match found for $toaddr"; }
    }
    else {
        $error_str = "Can't open $patfile";
    }
}

$recip = $error_recip if ($error_str);
unless (open(MAIL, "| $mail $recip")) {
    syslog(LOG_MAIL|LOG_ALERT, "forward mailer can't invoke sendmail!\n");
    die "forward mailer can't invoke sendmail!\n";
}

select(MAIL);
if ($error_str) {
    print <<"EOErrHead";
    To: $recip
    From: postmaster@netmarket.com (Forwarding Mailer)
    Reply-to: postmaster@netmarket.com
    Subject: ERROR: Forwarding mailer

    An error has occurred in the Forwarding mailer process. The error message
    generated is:

        $error_str

```

The original message follows:

=====

```
EOErrHead
}
while (<STDIN>) { print; }
close(MAIL);
exit(0);
```


Appendix B: Sample patterns File

```

####
#### beeper.netmarket.com -- people to annoy
####

(?:)^jim@beeper.netmarket.com$      beep-jxh@netmarket.com
(?:)^beeper.netmarket.com$          beep-$u@netmarket.com

####
#### infersys.com (personal domain for irilyth@netmarket.com)
####

(?:)^majordomo@infersys.com$      majordomo-at-infersys@netmarket.com
(?:)^majordomo-owner@infersys.com$ majordomo-owner-at-infersys@netmarket.com
(?:)^boston-b5@infersys.com$      boston-b5-at-infersys@netmarket.com
(?:)^owner-boston-b5@infersys.com$ owner-boston-b5-at-infersys@netmarket.com
(?:)^info@infersys.com$           info-at-infersys@netmarket.com
(?:)^irilyth@infersys.com$        irilyth@netmarket.com
(?:)^.*@infersys.com$            irilyth@infersys.com

####
#### ftd.com (Flowers and confused people at world.std.com)
####

(?:)^webmaster@ftd.com$           webmaster@netmarket.com
(?:)^postmaster@ftd.com$          postmaster@netmarket.com
(?:)^@ftd.com$                    $u@some-isp.net
(?:)^@world.ftd.com$              ftd-bounce-bot@netmarket.com

####
#### Some useful defaults to catch everything else
####

(?:)^(web|host|post)master@.*$    $u
(?:)^root@.*$                     root
(?:)^mailer-daemon@.*$            mailer-daemon
(?:)^.*$                          bounce-bot@netmarket.com

```

How to Get There From Here: Scaling the Enterprise-Wide Mail Infrastructure

Michael Grubb – Duke University

ABSTRACT

Recent explosions in volume of mail traffic and in numbers of people making use of Internet email has had a rapidly negative effect on many sites built with assumptions about low mail volume. Older solutions such as NFS are found not to scale upwards to the new email demands of the enterprise-wide email infrastructure. This paper examines this widely-experienced scenario as it unfolded at Duke University, discusses the problems experienced along the way, and describes in detail a solution for migration to a larger-scale infrastructural model that is incremental, transparent to users, and cost-effective.

The Way We Were

Like many installations, electronic mail handling at Duke began in earnest with Unix systems implementing widely-available and well-understood local mail delivery via a local filesystem, and using sendmail [Allman] for long-haul mail transport. When a distributed cluster system was installed for academic use, it was only natural to extend this paradigm using NFS [RFC1094] to provide remote access to a single mail spool for the whole set of machines, and leaving other elements of the local mail system in place.

This cluster mailhub arrangement [Cuccia] is a common scenario and appears to be functional for relatively small numbers of client machines, users, and messages delivered. This level of functionality is deceiving, however, because it does not scale upwards well.

The first problem area we noticed was in the time needed for directory name lookups and file reads of mailboxes via NFS. As the number of users grew, the number of files in the NFS-exported mail spool grew to the point that the server's directory name lookup cache was largely ineffective in preventing frequent sequential seeks for name information in the very large mail spool directory. This caused a slowdown in NFS accesses that caused increased load on the nfsd processes running on the mailhub machine.

At the same time, the number of incoming messages per day to the system was increasing dramatically. Each incoming message was handled by a forked sendmail child to handle the SMTP connection; at peak times (typically between noon and 4 pm local time on weekdays) the sheer number of incoming simultaneous SMTP connections, in conjunction with the increased load already being experienced by the nfsd processes, would overwhelm the available processing time, and the length of the pending run queue would grow precipitously. The resulting excessively high load average would be noticed by sendmail, which would then begin to refuse incoming

connections. The pending messages would be processed and the system load average would begin to fall, to the point that load was sufficiently low for sendmail once again to begin accepting incoming SMTP connections, and the cycle would start anew almost immediately.

It was clear in this scenario that the destabilizing factor was the spiking behavior caused by the unpredictable numbers of incoming SMTP connections. The solution adopted was to serialize incoming SMTP connections to the mailhub machine by setting up a mail gateway system. The MX records for the domain address in our BIND name server tables were altered so that all incoming connections would be routed to the mail gateway, like so:

```
acpub.duke.edu. IN MX
          9 gateway.acpub.duke.edu.
```

Sendmail on the mail gateway was configured to queue all messages. This can be done with the “-odq” command line option, accompanied by periodic queue runs with a separate invocation of sendmail. At our site, however, forced queuing of outgoing SMTP messages was done by setting the “e” (for “expensive”) flag on the SMTP mailer definitions in sendmail.cf, as in:

```
Msmtp, P=..., F=mDFMuXe, S=...,
        R=..., E=..., L=...,
        M=..., T=..., A=...
```

This can be done simply with one command using sendmail v8's m4 configuration facility:

```
define('SMTP_MAILER_FLAGS', 'e')
```

If the ‘c’ option (or “HoldExpensive” in sendmail 8.7) is set in the sendmail.cf of the mail gateway, then messages bound for delivery agents with the expensive flag set will be automatically enqueued and delivery attempted at the next queue run rather than immediately. This can be set using sendmail v8's m4 command:

```
define('confCON_EXPENSIVE', 'True')
```

The combination of the expensive flag on the SMTP delivery agents and the HoldExpensive option results in all SMTP messages being enqueued and delivered to the mailhub host serially. Frequent queue runs then safely pass messages to the mailhub without spiking load as specified at the command invocation, as in `"/usr/lib/sendmail -bd -q5m"` to run the queue every 5 minutes. Sendmail 8.7's `"MinQueueAge"` option allows such a system to be further configured with a minimum time in queue before delivery of a message is retried. We set `"MinQueueAge"` at our site to one hour so that down external sites would not clog up queue runs of critical messages to the mailhub.

In order to prevent mail on the mail gateway from looping back or from causing sendmail to notice the loop and bounce the message, a rule was added to ruleset 3 to automatically pass all recognized local messages to the mailhub (defined as \$H):

```
R$-      $: $1 < @ $H >
```

The mail gateway can be made to know about local users by carrying passwd table entries for each user; a dummy impossible password encryption such as `"nopass*"` prevents the user from using the system for other purposes. As long as the domain name is in class w in the mail gateway's `sendmail.cf` and the mail gateway knows about local users, the `sendmail.cf` rule above results in addresses of the form `"user@acpub.duke.edu"` being rewritten by the mail gateway as `"user@mailhub.acpub.duke.edu"`. A similar rule in ruleset 3 on the mailhub system rewrote addresses back to the form `"user@acpub.duke.edu"`:

```
R$* < @ $* . $M > $*  $: $1 < @ $M > $3
```

As long as the domain name is also in class w in the mailhub's `sendmail.cf` and the `sendmail v8` masquerading feature is used to set \$M, message headers are rewritten in such a way that users are blissfully shielded from the particulars of the mailhub/mailgateway arrangement. The domain name can be added to class w with a `"Cwacpub.duke.edu"` directive in the `sendmail.cf`, or by adding it to an `/etc/sendmail.cw` file pointed to by a `"Fw/etc/sendmail.cw"` directive in the `sendmail.cf`. The `sendmail v8` masquerading feature can be easily made use of with this m4 command:

```
MASQUERADE_AS(acpub.duke.edu)
```

We quickly learned that the mail gateway system was not able to keep up with its frequent queue runs due to the volume of incoming mail. This was evidenced by incoming messages remaining in the queue for longer than the queuing interval before being delivered to the mailhub. This would occur when messages came in at a rate faster than the mailhub could dequeue them

(roughly one message per second). Multiple mail gateways were then set up, and an MX round robin was created with multiple MX records at the same precedence level, in order to spread the incoming load across multiple systems; see Figure 1. We still found that roughly 10% of our incoming traffic continued to flow directly to the domain host from off-campus sites, presumably due to MTAs not handling MX records correctly, so we set the A record to a gateway system (gateway2, which received significantly less traffic over time than gateway1 and gateway3, presumably due to side-effects of the DNS round robin implementation) as well, as in:

```
acpub.duke.edu.      IN A 152.3.233.10
gateway2.acpub.duke.edu. IN A 152.3.233.10
```

The end result was that incoming SMTP connections went to one of several mail gateway systems, were enqueued, and were then passed on to the mailhub system in a small number of serialized connections rather than in the wild spikes of numbers of connections previously experienced by the mailhub.

This situation lasted stably for a short time until the continuing increase in mail usage began to overwhelm the mailhub. At that point, the system was experiencing peaks of ~300 simultaneous mail readers, there were more than 20,000 user accounts, and 120 client machines of the single NFS server, which was handling more than 1.5 million NFS accesses per day as measured by `nfsstat`. Disk I/O was abysmal due to the extreme contention among the `nfsd` processes (write wait times were in the dozens of seconds according to `iostat -x`). Users would wait for minutes between running a mail application and actually seeing their waiting messages.

The `syslog` facility was one of the few processes on the machine separable from the mail handling; the logging file system was moved to a new disk on a different SCSI interface in order to minimize I/O conflicts between the constant `syslogging` activity and the constant mail spool activity. A significant (> 10 second) improvement in disk access times resulted, but was not enough to be considered any kind of a solution.

At this point we were becoming desperate for any and all changes to make that would improve performance in any increment.¹ A number of kernel parameters on the mailhub system were manipulated for this purpose. On a Solaris system many kernel

¹[1] See [Cockroft] and [NFSTuning] for excellent discussions of the particulars of Solaris performance tuning.

```
acpub.duke.edu.  IN MX 9 gateway1.acpub.duke.edu.
                  IN MX 9 gateway2.acpub.duke.edu.
                  IN MX 9 gateway3.acpub.duke.edu.
```

Figure 1: Spreading incoming load

tables are sized automatically but can be increased to take advantage of available real memory by increasing the `maxusers` kernel parameter. We also maximally increased the size of the kernel's inode and buffer caches for the same purpose. On a Solaris 2.3 system (the exact numbers to use vary from version to version of Solaris), the `/etc/system` entries looked like this:

```
set maxusers=1024
set ufs_ninode=34906
set ncsiz=34906
set bufhwm=10240
```

NFS mount options on the client hosts were also manipulated in order to maximize efficiency of the NFS connections. In order to minimize storms of NFS retransmissions from the clients, the NFS request timeout and attribute caching periods were dramatically increased. [Stern] An `/etc/vfstab` entry to accomplish this for Solaris 2.3 systems looked like this:

```
mailhub:/var/mail - /var/mail \
nfs - yes bg,hard,intr,timeo=90,\
acregmin=60,acregmax=300,\
acdirmin=300,acdirmax=600
```

Some of these rather extreme values only make sense because the filesystem was a mail spool and thus directory and file attributes were changing very little. Don't try this with an ordinary multipurpose NFS-mounted file system!

Ultimately, the system ground to a halt. The active attempt of Solaris' `rpc.statd` to honor file locking via NFS and the impact of 20 or more NFS operations per second imposed such load on the server and introduced such instability, that the `nfsd` processes under extreme load experienced total meltdown. The system froze, and in a horrible set of experiences we described as "infocalypse", was unable to be restarted with locking honored between server and clients. The restarted server would receive storms of `rpc.statd` requests when it became available, would be unable to handle the load, and would die or freeze. On these occasions it was necessary to turn off NFS on the entire distributed system manually by killing `lockd` and `statd` processes on all client machines in order to bring the server back up stably, when NFS exporting could once again be turned on. This sort of experience is time-consuming and extremely disruptive to a busy system, and resulted in demands from users and management alike for a different way of doing things.

The Way We Wanted to Be

We knew where we wanted to wind up in our changes to the mail system, a system that had inadvertently become the central campus-wide email infrastructure. We were victims of our own success; the stable and fast system we had created for 5,000 users attracted 15,000 more users, who created load beyond the ability of the system to handle. By tweaking parameters on the mailhub, investing in larger and

more powerful mailhub hardware, and imposing mail gateways to handle incoming SMTP connections, we had grown the system as far as it could go with the present architecture.

Our desire was to have a truly scalable system that could keep up with future growth for a number of years. We wanted to replace mailbox access via NFS with the POP and IMAP protocols.

POP and IMAP are Internet protocols for remote mail access. POP (current version POP3 [RFC1725]), which has been around in various forms for much more than a decade, is very stable and widely implemented. POP provides access to a single remote mailbox. IMAP, which is somewhat newer and less widely-implemented, is much more flexible. IMAP provides access to multiple remote folders, as well as to individual messages within those folders, as opposed to the bulk mailbox operation of POP.

IMAP2 [RFC1176] (with extensions referred to as "IMAP2bis" [Crispin]) is currently the most widely-used version of the IMAP protocol, although implementations of IMAP4 [RFC1730] clients and servers now becoming available provide greatly expanded and long-awaited functionality such as shared folders and per folder access control lists. IMAP4 servers can also handle IMAP2bis requests from older clients for backward-compatibility. [RFC1732] IMAP4 server implementations typically also provide a daemon for POP3 access to mailboxes for additional backward-compatibility with the already-installed base of mail client software.

We wanted our new system to be able to make use of as many post office machines (that is, machines storing users' mailboxes) as necessary to handle load. Our desired architecture should be scalable to thousands of hosts and millions of messages per day. It should provide an infrastructure that accommodates mobile and disconnected operation. Points of failure should be spread out rather than concentrated in a single machine, as long as mean time between failures can be kept low. The new system should continue to work with the client hardware and software already in place, and it should provide additional functionality such as bulletin boards, shared mailboxes, and mailbox access control lists (ACLs). The IMAP4 protocol looked like a good candidate to provide the backwards compatibility (via widely-used IMAP2bis clients such as pine) and the new features we were looking for.

With a clear idea of what was wrong with our system and a clear idea of what new system we wanted to put in place, we were still paralyzed by the problem of "You can't get there from here." There was no clear, documented, or well-understood path for migrating from the one paradigm to the other. One option for migration appeared to be shutting down our mail systems for an extended downtime, then coming back up with the new servers in place. This option was deemed unacceptable for mission-critical mail

services. Another option was to set up the new system in parallel to the old system and offer users the choice of moving their mail to the new system with its added functionality. This option was deemed unacceptable because it would have been costly to implement and maintain two mail systems in parallel for an extended period of time, and because the user community was for the most part not technically inclined, costly to support, and not willing to abandon an already-familiar configuration, even when it was failing under load. The decision was made in this environment to attempt to perform the miracle of salvaging the existing system by converting it to the new paradigm without significantly interrupting mission-critical mail services.

How to Get There From Here

One key insight that made our migration possible was to implement multiple post office machines while still using NFS for mailbox access. This made it possible to move to a multiple post office architecture independently of other changes to the mail system.

The second key insight that allowed us to "get there from here" was the creation of a large number of CNAME DNS records that would allow users to configure their mail software to access their post office by a single unchanging name related to their userid, without having to know or care which post office was actually handling their mail. The most flexible way to do this is to set up a separate domain for your post offices, such as "mail.domain.com", and to have a CNAME for each userid, such as "user1.mail.domain.com", "user2.domain.mail.com", etc. This allows mailboxes to be rearranged on post office servers individually, and this large number of CNAME records is certainly workable on today's BIND name servers. Due to purely political considerations, a slightly different approach was followed at our site, using the same theory. CNAME records were created for each two-letter combination at the beginning of a userid. Thus, for a userid "hiro", his mail would no longer necessarily be available from "acpub.duke.edu", but would be guaranteed to be available from his post office, which would be "mail-hi.acpub.duke.edu", "h" and "i" being the first two letters of his userid. With these post office CNAMEs in place, it became possible to make changes to mail handling for small groups of users at a time without having an effect on all users simultaneously.

The overall migration plan was fairly simple²:

1. Create the mail-xx CNAMEs. Route mail from mail gateways to correct post offices using the CNAMEs and the sendmail v8 userdb.
2. Set up parallel post offices, exporting mail spools via NFS. Move groups of users to new post offices to spread load. Leave symbolic

links from original mail spool to allow old mail clients to continue to work. For each group moved, update mail-xx CNAMEs accordingly.

3. Turn on POP3 and IMAP2bis servers on each post office machine. Provide users with clients that access mail via POP or IMAP.
4. Turn off NFS exporting from post offices.
5. Set up IMAP4/POP3 servers, and move users group-by-group to the new post offices. Convert vacated post offices to the new IMAP4/POP3 setup, and continue by moving additional mailboxes to the new post offices, repeating until all mailboxes and post offices are converted.

The beauty of this migration path is that each step is completely incremental. It is possible to take each step as slowly or as quickly as necessary, and it is possible to back out of each step along the way. There are no "flag days" for users, no single events of system-wide change that require all users to alter their behavior – most changes are transparent to end users, and those that are not can be easily coordinated with users on an individual basis.

From the point of view of the users, there is a single change: the configuration of their mail reading software must be updated to access their mailbox via the new mail-xx CNAME. Once NFS access to mailboxes is turned off on the post offices, POP and IMAP clients that haven't been updated will no longer be able to access a user's mailbox, although mail delivery will continue uninterrupted. This change can be announced as far in advance and as repeatedly as necessary. By following the logs of POP and IMAP accesses, it is possible to determine if necessary those users who have not reconfigured their clients. These users can then be contacted individually for assistance in reconfiguration. The deadline can be moved back as many times as necessary to avoid seriously inconveniencing any users.

The IMAP4/POP3 server software we decided to use was the Cyrus³ mail software from Carnegie-Mellon University. This software suite was designed for maximum efficiency for very large sites like ours with dedicated, secured mail servers. Use of this software would minimize the load created by the `imapd` processes handling client IMAP connections because single messages are loaded into memory rather than entire mailboxes⁴. However, switching to use of the Cyrus software introduced a complication to our migration path. Since the Cyrus server uses a different

³The Cyrus Project at Carnegie-Mellon University is described in detail at <http://andrew2.andrew.cmu.edu/cyrus/cyrus.html>.

⁴This is also possible with the University of Washington `imapd` using a non-Berkeley mailbox format; however, this solution presents the same problem of mailbox conversion without gaining the security and administrative simplicity of a sealed server that doesn't know about user accounts.

²The original design of this migration plan grew out of a series of conversations between Karl Ramm and the author.

format for storing mailboxes on disk it was necessary to create new Cyrus servers and convert mailboxes to the new format as they were moved from old servers to the new servers.

Proper sizing of the post office servers during and after this migration is critical to maintaining continuous highly-reliable mail service. One large difference between most users accessing mailboxes via NFS and most users accessing the same Berkeley-style mailboxes via an IMAP2 server is in memory utilization: one or two copies of the entire mailbox are loaded into memory while the `imapd` is running. The habit of some users of accumulating very large mailboxes, combined with another common habit of opening a mail reader (and thus an `imapd` on the post office) and leaving it open for a long period of time, can result in very large memory requirements on the post offices. In our environment 512MB of RAM and five times that amount in swap space was barely sufficient to prevent some crashes at peak times. Luckily the memory requirements decrease after the conversion is completed due to the post-conversion IMAP daemon being able to access mailbox messages individually.

Once the IMAP4 servers are in place it is possible to begin making new clients available to users that take advantage of the new features of this protocol. IMAP2 and IMAP2bis clients should remain minimally functional with the new servers, although they will not provide access to the expanded IMAP4 capabilities such as shared message folders, per-folder access control lists, resynchronization for disconnected clients, and so forth.

Once the migration is complete, users have configured their POP and IMAP clients to access their mailboxes using the `mail-xx` CNAMEs. An IMSP [Collyer] service is also made available for clients that can make use of that service for mailbox location discovery. IMSP is a deprecated protocol for remote authenticated service of mail client configuration information such as mailbox location and addressbook information. IMSP is currently being reworked by the IETF working group on ACAP [Wall], which is more broadly targeted to supply configuration information for other types of clients as well. It is to be hoped that ACAP implementations will continue to provide the mailbox discovery mechanism provided by the IMSP server.

Conclusions

This paper details one odyssey from a small mail system to an enterprise-wide mission-critical infrastructure. There are many ways to tackle this problem, thanks in large part to the power and flexibility of `sendmail` as a mail transport agent. Some of the particular examples given here are very specific to our primary operating system at the time, Solaris 2.3, but the principles at stake should carry over to other Unix

implementations. By providing what we did not have when embarking on this project, one clearly-documented example of migration from a small to a large-sized mail system, it is to be hoped that this paper helps at least one other site to avoid some of the same mistakes. It is possible to "get there from here", but it is best to plan ahead carefully rather than to wait for infocalyptic disasters to occur and push you forward.

Author Information

Michael Grubb is a Senior Systems Programmer for Duke University's Office of Information Technology. He is also a licensed attorney. He can be reached via email at mg@duke.edu or via post at Box 90132, Durham NC 27708-0132 USA.

References

- [Albitz] Albitz, Paul, and Cricket Liu. *DNS and BIND*. Sebastopol, CA: O'Reilly & Associates, Inc., 1992.
- [Allman] Allman, Eric. "Sendmail, An Internetwork Mail Router" in the *BSD Unix Documentation Set*, Berkeley, CA: University of California, 1986-1993.
- [Avolio] Avolio, Frederick M., and Paul A. Vixie. *Sendmail: Theory and Practice*. Boston, MA: Digital Press, 1995.
- [Cockroft] Cockroft, Adrian. *Sun Performance and Tuning*. Mountain View, CA: SunSoft Press, 1995.
- [Collyer] Collyer, Wallace. *IMSP - Internet Email Scales to the Enterprise*. Available from <http://andrew2.andrew.cmu.edu/cyrus/impsp/impsp-white.html>.
- [Costales] Costales, Bryan, with Eric Allman and Neil Rickert. *sendmail*. Sebastopol, CA: O'Reilly & Associates, Inc., 1993.
- [Crispin] Crispin, M. IMAP2bis: "Extensions to the IMAP2 Protocol, 1992". Available from <ftp://ftp.cac.washington.edu/mail/IMAP2bis.TXT>.
- [Cuccia] Cuccia, Nichlos H. "The Design and Implementation of a Multihub Electronic Mail Environment". San Diego, CA: *USENIX Proceedings - Lisa V*; October 3, 1991.
- [Darmohray] Darmohray, Tina M. "A `sendmail.cf` Scheme for a Large Network". San Diego, CA: *USENIX Proceedings - Lisa V*; October 3, 1991.
- [Harrison] Harrison, Helen E. "A Domain Mail System on Dissimilar Computers: Trials and Tribulations of SMTP". Colorado Springs, CO: *USENIX Proceedings - LISA IV*; October 19, 1990.
- [Myers] Myers, J. G. "IMSP - Internet Message Support Protocol", 1995. Available from <http://andrew2.andrew.cmu.edu/cyrus/rfc/impsp.html>.
- [Nemeth] Nemeth, Evi, Garth Snyder, Scott Seebass, and Trent R. Hein. "Electronic Mail", Chapter

- 21 in *UNIX System Administration Handbook*, 2d ed. Englewood Cliffs, NJ: Prentice Hall, 1995.
- [NFSTuning] Sun Microsystems. *SMCC NFS Server Performance and Tuning Guide*. Mountain View, CA, 1994.
- [RFC821] Postel, Jonathan B. *RFC 821: Simple Mail Transfer Protocol*, 1982.
- [RFC822] Crocker, David H. *RFC 822: Standard for the Format of ARPA Internet Text Messages*, 1982.
- [RFC974] Partridge, Craig. *RFC 974: Mail Routing and the Domain System*, 1986.
- [RFC1094] Sun Microsystems, Inc. *RFC 1094: NFS: Network File System Protocol Specification*, 1989.
- [RFC1123] Braden, R., ed. *RFC 1123: Requirements for Internet Hosts – Application and Support*, 1989.
- [RFC1176] Crispin, M. *RFC 1176: Interactive Mail Access Protocol – Version 2*, 1990.
- [RFC1725] Myers, J. and M. Rose. *RFC 1725: Post Office Protocol – Version 3*, 1994.
- [RFC1730] Crispin, M. *RFC 1730: Internet Message Access Protocol – Version 4*, 1994.
- [RFC1732] Crispin, M. *RFC 1732: IMAP4 Compatibility with IMAP2 and IMAP2bis*, 1994.
- [Stern] Stern, Hal. *Managing NFS and NIS*. Sebastopol, CA: O'Reilly & Associates, Inc., 1991.
- [Wall] Wall, Matthew. *The Application Configuration Access Protocol and User Mobility on the Internet*, 1996. Available from <http://andrew2.andrew.cmu.edu/cyrus/acap/acap-white-paper.html>.

Appendix A: Pictorial Diagrams
Mail infrastructure at Duke - after

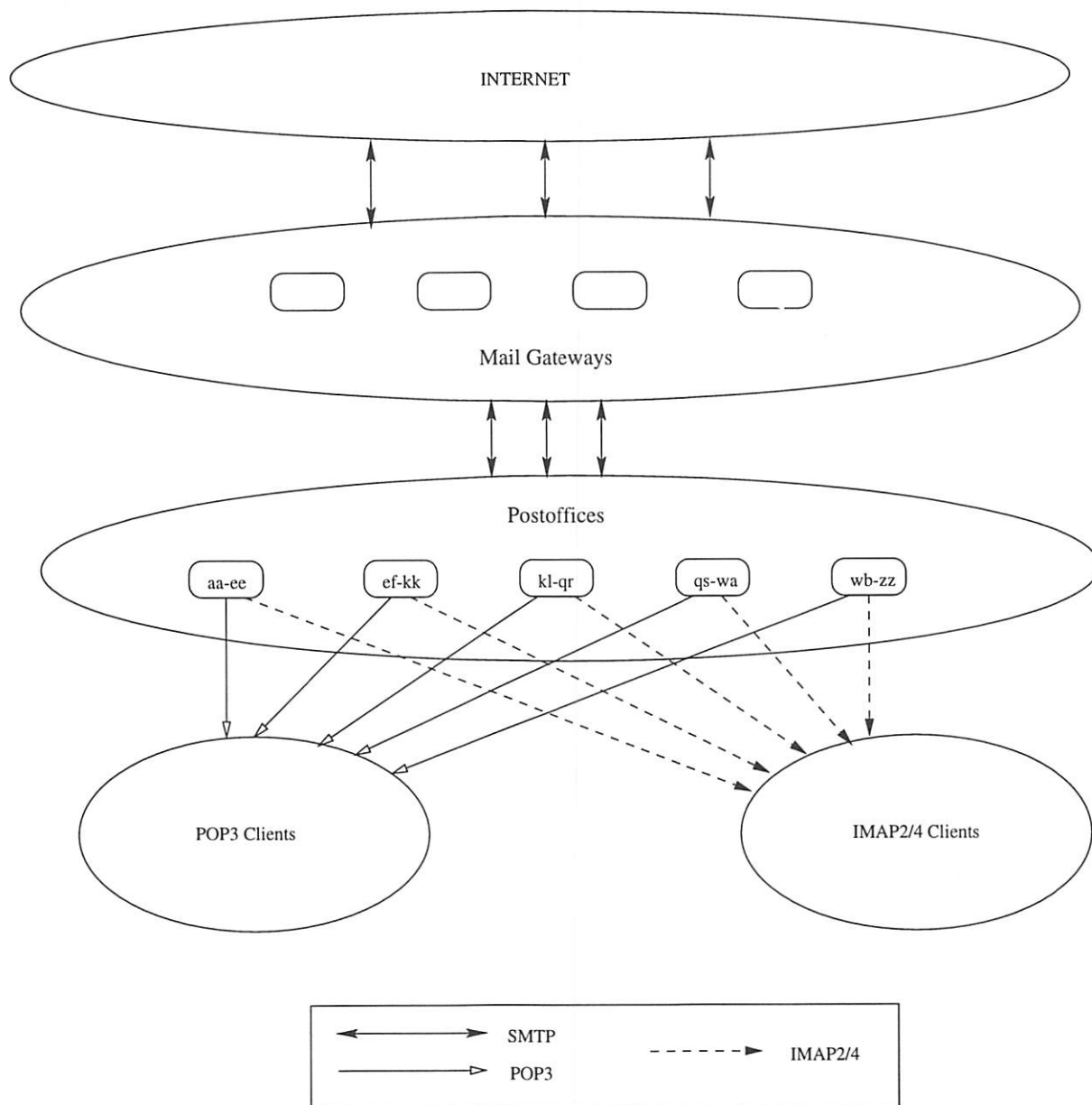


Figure 1: After the Scaling

Mail infrastructure at Duke - migration

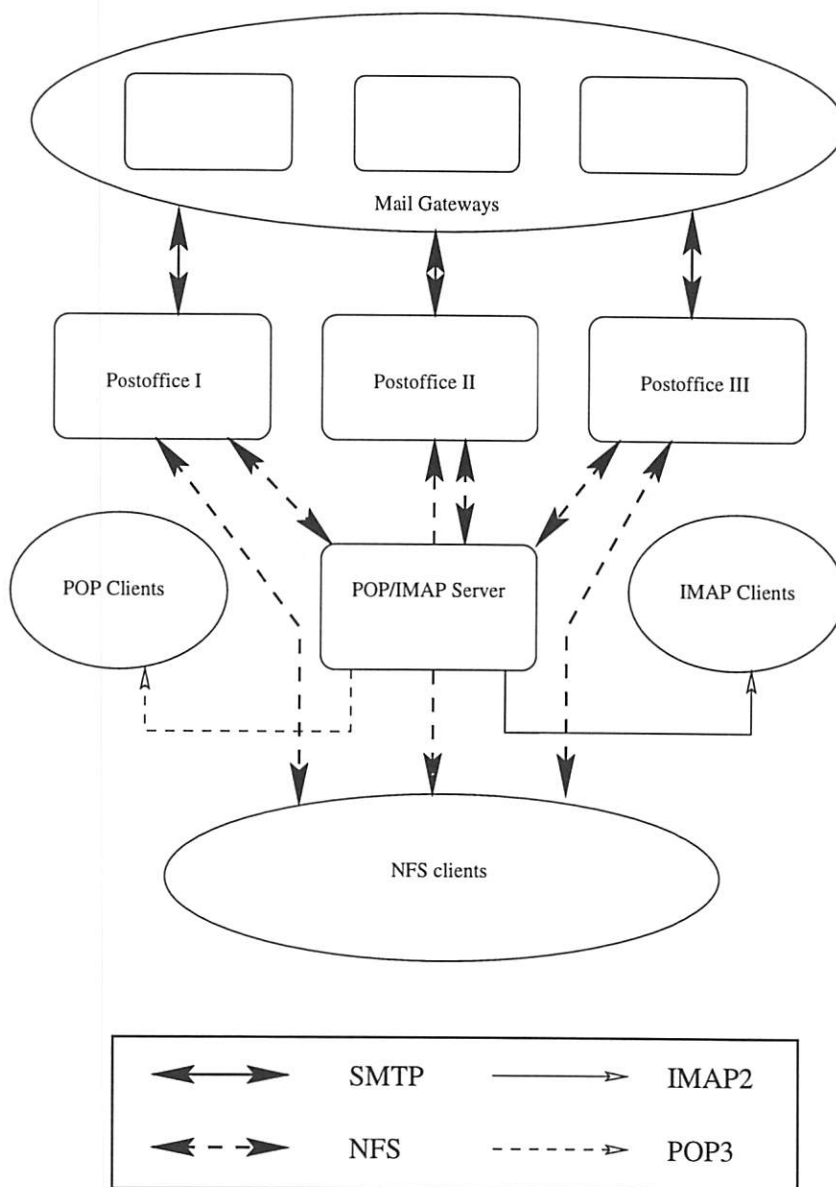


Figure 2: Migration

Automatic and Reliable Elimination of E-mail Loops Based on Statistical Analysis

E. Solana, V. Baggiolini, M. Ramluckun and J. Harms
– University of Geneva (Switzerland)

ABSTRACT

One of the major threats to the E-mail service are message loops, mail messages which endlessly circulate between a number of E-mail relays and severely deteriorate or even interrupt the messaging service. Although the standards ruling E-mail communications include mechanisms to prevent message loops, these mechanisms are often unimplemented and, if they are, can fail to work reliably under certain circumstances. This paper describes a novel approach to the automatic and reliable elimination of looping messages. Statistical properties of the message traffic are analyzed to detect messages that might be looping. Such messages are earmarked, and if they are observed to actually be looping, they are eliminated. This method has been implemented and successfully applied in the multi-protocol E-mail environment of the University of Geneva.

Problem description

A message is said to be looping if it is forwarded endlessly in a circular manner among a set of *mail relays* (*Message Transfer Agents* or *MTAs*). Normally, two or three mail relays take part in a message loop, but there can be more than that. The period of a loop (the time it takes to complete) ranges from a few seconds to a few minutes. Looping messages usually grow in size, since mail relays typically add header lines to messages they process.

A message that loops during a weekend can be forwarded tens of thousands of times, growing in size to megabytes and producing megabytes of logging data. The situation is even worse if *distribution lists* are involved (c.f. [1]): the number of messages explodes and soon the mail system crashes under the load. These scenarios illustrate how important it is to detect and eliminate looping messages *automatically*.

Loops are caused by wrong addressing configuration on one or more systems:

- *bad configuration* of mail relays, e.g., a relay that does not “recognize” messages addressed to it and, instead of keeping them for local delivery, forwards them to a central mail relay which sends them back again¹.
- *automatic forward mechanisms* such as aliases, distribution lists, or personal forward mechanisms (*.forward* files) pointing to invalid addresses.
- *automatic reply mechanisms*, such as vacation programs or non-delivery reports.

The mechanisms that are directly under control of a proficient administrator account for only a small part of the errors. The problem lies much more in the

forwarding facilities that are under user control, such as *.forward* files and vacation programs or even mail relays installed without the administrator knowing. Furthermore, even if all the mail relays in a given responsibility domain are perfectly configured, a mail transaction with a misconfigured or non-compliant site may result in a message loop.

Standard Mechanisms For Loop Detection

There are two typical countermeasures that are integrated into the mail relays or taken by postmasters:

- *Counting the Received from lines* in the message header. RFC 821 [2] says every mail relay processing a message must add such a line to the message header and count the number of *Received from* lines already contained. When the header encloses more than a pre-defined number (e.g., 25) of such lines, the message is considered to be looping and automatically sent back to the expeditor together with a non-delivery report. Fixing such a limit is quite a delicate task (especially due to the growing complexity of the Internet).
- *Tracing Message IDs*. Message IDs allow to uniquely identify E-mail messages. Postmasters can inspect the log-files and check if the same Message ID recurs multiple times. If this is the case, the corresponding message is considered to be looping.

Both of these approaches work well if there are only mail relays of one protocol involved. However, they can fail in a multi-protocol environment (e.g., SMTP [2, 3] together with X.400 [4]²) or in case of

¹This is the behavior of many pre-installed *sendmails*.

²Although the standard for protocol conversion (RFC-1327 [5]) describes how trace fields should be converted, a number of mail relay products are not compliant to these recommendations.

faulty configurations or non-compliance to the protocol recommendations. When a message is translated into a different E-mail protocol by a gateway, *Received from* lines can get lost; some gateways can even be explicitly configured to eliminate them. The same problem exists for Message IDs: some mail relays change the old Message ID and assign a new one. Furthermore, both methods fail if a new message is generated in the looping process (e.g., vacation or non-delivery reports).

Related work

Most existing solutions rely on the mechanisms described above and are normally included in mail relay implementations. A recent work being conducted by D. Bernstein [7] proposes a set of recommendations for so-called *automailers* (a generic term designating mail relays, list exploders, vacation programs, etc.). Due to the great number and diversity of existent software implementations, compliance to these recommendations will not be achieved in a foreseeable future (if ever...) and the need of external mechanisms, as the one proposed in this paper, will persist.

To our knowledge, there is no independent, commonly agreed-on solution for preventing loops between E-mail systems. The approach adopted by some administrators consists in periodically observing the message traffic for a given period of time in order to find potential loops.

Unfortunately, the most common practice consists in reacting "a posteriori", alerted by the unpleasant effects of a loop in the E-mail system.

A more scientific approach to solve the problem of looping messages has been studied by J. F. Paccini in his Ph.D. thesis [8]. He proposes a method to identify a loop by tracing suspect messages amongst the set of involved mail relays, the tracing information being shared by each mail relay by means of a standardized distributed database (X.500, [9]). The two major drawbacks of this approach are the significant management overhead introduced by a distributed database and the security issues related to the sharing of sensitive information (message trace statistics) across administrative domain borders.

Our Approach

The proposed method is based on a *statistical analysis* of the messages recently exchanged by a single mail relay. As will be explained, message loops have very particular statistical properties that are visible at a *single observation point*. We propose to exploit these properties in order to detect suspect messages without having to consider management information from other mail relays.

Since message loops occur occasionally, we have chosen to trigger the detection mechanism upon specific conditions directly related to loop problems: *abnormal message rate, disk, and/or CPU usage*.

E-mail related information can be obtained by periodically querying the mail relay console or by means of an SNMP agent that provides monitoring data in the form of a MIB (c.f. [6]). Information concerning system resources can be retrieved by means of standard UNIX commands.

The process of loop detection and suppression is composed of two phases: the first focuses on the *detection of suspect messages* based on the logging information of recent message exchanges and the second aims at the *accurate identification and elimination of messages* that do actually loop.

Detection Of Suspect Messages

The aim of this process is to identify those messages that might be looping. To achieve this, we build a *statistical database* based on the historical information (log-files) generated by the mail relay where the observation point is located. Note that the necessary data is contained in the log-files of about any existing E-mail implementation:

- *The Sender and Receiver addresses.* Since Message ID information is not always reliable, the only invariant of a looping message observed from one specific point are the sender and receiver addresses of the envelope. Figure 3 shows that this assertion does not hold when the observation point changes (address rewritten by involved mail relays). This information, that we denote *SR pair*, constitutes the *search key* of our statistical database.
- *The number of messages exchanged by this SR pair.*
- *Absolute time of last message exchange.* This value is used to determine whether the loop is still active or not.
- *Time lapses between occurrences of the same SR pair:* A set of integer values containing the time intervals in seconds between messages with identical *SR pairs*.

Based on this, we pick the most often recurring *SR pairs* for further analysis. Generally, this is a threshold-based decision which depends on site characteristics and the extent of historical information being considered. If the loop has been running for some time, the number of messages exchanged might be sufficient to identify a potential loop³.

However, if the loop has just begun and only a few iterations have occurred, a more elaborate analysis is required to more accurately distinguish looping messages from normal traffic. This is done by performing *statistical computations* over the time lapses set, namely:

- *Mean value of time lapses:* The arithmetic mean value calculated over the above set of

³3000 messages are unlikely to be exchanged by two partners in a single day.

message lapses. This value represents the frequency of arrival of suspect messages. A low mean value implies a high frequency of arrival of messages with identical *SR pairs* and, thus, a high probability of a loop.

- *Variance* of time lapses: The variance of time lapses measures the deviation from the mean value. This value is very relevant since looping messages are normally processed at uniform time intervals and, as a consequence, display a

remarkably low variance. On the contrary, normal E-mail traffic has a very irregular time distribution resulting in very high variance values.

Based on this statistical information, we have elaborated a *heuristic function* for computing the *loop probability* for each *SR pair* appearing repeatedly in the log-files. Figure 1 shows the inner workings of the heuristic function that has been used to detect loops in our site.

```

/*
Variables for each SR Pair:
    number_of_messages, time_of_last_message, lapses_variance,
    lapses_mean_value;
Thresholds (depending on site characteristics):
    minimum_amount_of_messages, timefactor,
    suspect_var_threshold, suspect_mean_value
*/

function loop_probability_fct {
    loop_probability = 0; /* initial value */
    /* if the number of messages is under a given threshold don't go
       any further. (statistical computations are not sufficiently
       accurate for a small amount of msgs.) */
    if (number_of_messages < minimum_amount_of_messages)
        return(loop_probability);
    if (((current_time - time_of_last_message) / mean_value)
        > timefactor)

    /* if the time interval between the last message and the current time is
       timefactor times higher than the mean value, we assume that the
       potential loop has been fixed */
        return(loop_probability);
    /* if the variance is below a given threshold, increase probability */
    if (lapses_variance < suspect_var_threshold)
        loop_probability += 1 - lapses_variance / suspect_var_threshold;
    /* ditto with the mean value */
    if (lapses_mean_value < suspect_mean_value)
        loop_probability += 1 - lapses_mean_value / suspect_mean_value;
    if (number_of_messages > suspect_number_of_messages)
    /* if the number of messages exchanged is abnormally high, increase
       probability */
        loop_probability += number_of_messages /
            total_number_of_exchanged_messages;
    if (loop_probability > 1)
        return (1);
    else
        return(loop_probability);
} /* loop_probability_fct */

```

Figure 1: Heuristic Function Computing Loop Probability

Firstly, the function tests if the number of messages exchanged is sufficiently high to ensure accurate statistical computations. The lapse between the last message and the current time is compared to the mean value of previous transactions lapses in order to detect if the potential loop is still active. If both tests yield positive results, the effective loop probability is computed. Low values for either the variance or the mean value increase loop probability. An abnormally high number of messages exchanged will also augment the result returned by this function. It should be noted that a suspect value in any of these parameters will have significant impact in the overall loop probability returned by the function.

The following example shows the results of the loop probability computations for a message loop (Figure 2a) and for a regular group of messages exchanged by an *SR pair* (Figure 2b). In the first case, note both the high frequency (approx. one message every 20 seconds) and the uniform distribution of time lapses between message arrival times.

This results in very low mean and variance values and, as a consequence, in a very high (close to 1) loop probability value. In the second example, where normal mail traffic is analyzed, the high variance and mean values yield a remarkably low loop probability.

Figure 3 shows a typical loop scenario where a message (addressed to `loop@mail1[23]`) circulates endlessly between three mail relays (for instance, due to an incorrect forward address combination for the loop user). The statistics are retrieved at the observation point number 1.

Note that due to the high number of messages exchanged and the low mean and variance values, the loop probability (`loopprob`) for this message is

very high compared to the remaining messages. As a consequence, the *SR pair* [`a@mail3` | `loop@mail1`] is considered to be suspect and used as input for the next processing stage.

```
MESSAGES FROM solana@cui.unige.ch
TO = loop@bad.domain
NUMBER OF MESSAGES EXCHANGED = 75
TIMELAST = 05/04 10:29:55
LAPSES = [17, 17, 32, 18, 17, 18,
          31, 33, 18, 19, 17, 17,
          17, 34, 17, 23, 20, 22,
          17, 18, 17, 20, ...]
```

```
MEAN VALUE = 20
VARIANCE = 66
LOOP PROBABILITY = 0.97
```

Figure 2a: Statistical Properties of a Loop

```
MESSAGES FROM solana@cui.unige.ch
TO normal@cui.unige.ch
NUMBER OF MESSAGES EXCHANGED = 70
TIMELAST = 05/04 10:25:53
LAPSES = [205, 1800, 3357, 264, 45,
          188, 256, 313, 294, 141,
          107, 135, 161, 689, 1267,
          691, 5402, 24640, ...]
```

```
MEAN VALUE = 1125
VARIANCE = 85443
LOOP PROBABILITY = 0.01
```

Figure 2b: Statistical Properties of Normal Messages

The next phase will permit to accurately determine if the selected *SR pairs* actually correspond to a message loop. If yes, the necessary corrective actions are carried out.

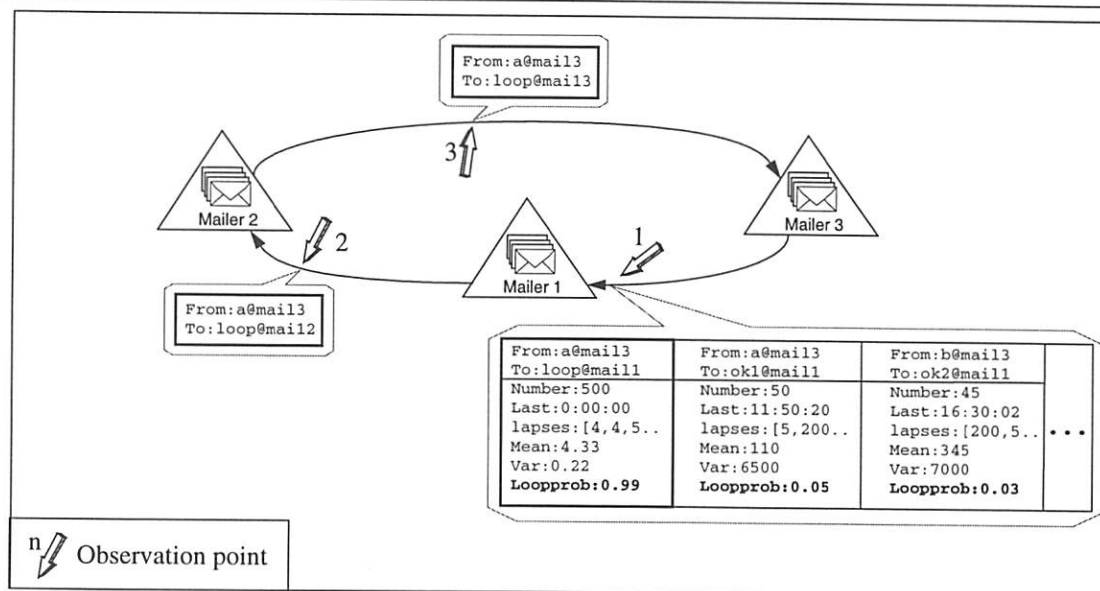


Figure 3: Detection of Suspect Messages

Identification and Elimination of a Message Loop

If suspicious *SR pairs* have been identified, the program will actively wait for the corresponding messages to arrive and carry out the following actions for each of them:

- *Freeze the suspect incoming message* in order to enable write operations on the message header.
- *Mark the message with an Unique Identification Tag (UIT)* using a reserved *header record*⁴ and a well-defined string (containing, for instance, the mail relay identification plus an exact time value).
The UIT of each suspicious *SR pair* will be recorded for further identification.
- *Flush the message*, i.e., force the message to leave the mail relay again.
- *Detect the tagged message*. Wait for the tagged message to arrive again. The arrival of a message containing both a suspicious *SR pair* and an UIT means that the message has been processed twice by the mail relay with identical envelope addresses. This fact is an unequivocal loop symptom. If the envelope addresses were different, the presence of an UIT alone would not constitute a sufficient condition for this assertion since regular messages might be processed twice by a mail relay as a result of a user forward to an external address⁵.
- *Remove the message loop*. The immediate action to eliminate a message loop may consist in "bouncing" the message back to the sender with an error diagnostic or to discard the message and generate a report directed to the local E-mail administrator. Further actions must then be undertaken to diagnose and fix the actual reasons (configuration errors, protocol failures, etc.) of the loop.

Implementation Experience

The proposed loop detection and removal mechanism has been fully implemented in two central mail relays of the operational E-mail service in the University of Geneva: the main gateway between X.400 and SMTP and an SMTP relay containing user mailboxes. Apart from a few message loops that were introduced for testing, both mail relays are subject to normal operating conditions.

During a short calibration period, parameters of the heuristic function were tuned to the particular traffic. After this period, the system could distinguish with remarkable accuracy looping messages from

regular traffic. When combined with the *mark and remove* mechanism described above, the system automatically eliminate all possible message loops.

The loop detection script has been implemented in PERL and is currently used by postmasters of the Geneva University network and by SWITCH, the major Swiss Internet service provider for academia. The package is freely available at <http://cuiwww.unige.ch/~solana/loop-watch.tar>.

This work is part of a larger project⁶ that focuses on distributed application management with special focus on E-mail management.

Conclusions

During several months of utilization, the loop finder script has automatically detected and removed several message loops that in the past would have remained undetected for several days and might have caused serious problems to the E-mail service. This shows that even the analysis of basic statistical parameters is sufficient to distinguish looping messages from ordinary traffic.

As described in the previous section, the loop detection process obtains the message informations from the log-files of mail relays implementations. A version that processes the information provided by an *SMTP sniffer* – a program capable of analyzing SMTP transactions obtained directly "on the wire" – is being finalized. This will notably enhance the portability (no need to consider proprietary log-file formats) and the reliability (no influence from mail relay software errors).

Although the main objective of our work consisted in detecting and removing message loops, the accurate statistic information obtained, has brought significant insight into the E-mail traffic conditions of our site. In particular, relying on this information, we have operated important changes in the topology of our campus E-mail architecture resulting in an improved load balancing among message relays. Furthermore, the nature of information provided by this script can be easily exploited for accounting and billing issues.

Experience shows that in widely distributed and heterogeneous environments such as electronic mail, the recommendations of standard protocols are often not correctly implemented. Mail loops are a good example of a problem that theoretically should not happen but practically does. Furthermore, the incredible growth of E-mail utilization will certainly aggravate this problem. Techniques as the one described in this paper can detect malfunctions before the quality of service provided to end-users degrades.

⁴The header record should be carefully chosen to allow multi-protocol inter-operability.

⁵As a consequence of the address rewriting inherent to the forward process, the message enters and leaves the mail relay with different recipient addresses.

⁶Project funded by the "Fonds National suisse pour la Recherche Scientifique" (FNRS).

Author Information

Eduardo Solana received his M.Sc. degree in Computer Science in 1991 from the University of Geneva. He has been a member of the IBM technical staff for three years. Currently he is a Ph.D. student in the Teleinformatics and Operating Systems (TIOS) group of the University of Geneva. His interests are in the fields of Computer Communications, Management of Distributed Applications, Network Security and Cryptography. Write to him via e-mail at <Eduardo.Solana@cui.unige.ch>.

Vito Baggiolini is a Ph.D. student at the University of Geneva where he has been working since 1993 on the management of distributed applications. He graduated as an electronic engineer from the Federal Institute of Technology in Zurich in 1989. Between 1990 and 1993 he worked at the European Space Agency in Frascati (Italy) in the field of computer networking.

Mira Ramluckun received her M.Sc. degree in Computer Science from the University of Compiègne (Paris). Between 1992 and 1994 she worked in the Linguistics Department of the University of Geneva in the field of Natural Language Processing. Since 1994 she is a Ph.D. student in the Teleinformatics and Operating Systems (TIOS) group of the University of Geneva. Her interests are in the fields of Computer Communications, Database Design and Natural Language Processing.

Jürgen Harms. Studies in physics at the Polytechnic School of Vienna (Austria) and in electronic engineering at the University of California (USA). Since 1972 professor of computer science at the University of Geneva where he heads the computer department. President of the SWITCH foundation (Swiss national research network). Fields of interest: Operating Systems, Teleinformatics.

Bibliography

- [1] J. Udell. "E-mail adventures". *BYTE* magazine, Volume 21, Number 4. April, 1996.
- [2] J. Postel. "Simple Mail Transfer Protocol". *Request for Comments 821*. August, 1982.
- [3] D. Crocker. "Standard for the format of ARPA Internet text messages". *Request for Comments 822*. August, 1982.
- [4] ITU and CIITT – *Data Communications Networks*. Message Handling Systems. Recommendations X.400-X.420, 1988.
- [5] S. Hardcastle-Kille. "Mapping between X.400 (1988) / ISO 10021 and RFC 822". *Request for Comments 1327*. May, 1992.
- [6] S. Kille, N. Freed: "Mail Monitoring MIB", *Request for Comments 1566*. January 1994.
- [7] D. Bernstein. *Tools in the War on Mail Loops*. Internet-draft: draft-bernstein-mail-loops-war-00.txt. April, 1996.

[8] J. F. Paccini. *De la gestion de réseaux à la gestion d'applications: le rôle des données historiques*. Ph. D. Thesis, Université de Genève, 1994.

[9] ITU and CIITT – *Data Communications Networks. Directory*. Recommendations X.500-X.521, 1988.

MajorCool: A Web Interface To Majordomo

Bill Houle – NCR Corporation

ABSTRACT

Majordomo is a popular mailing list manager and auto-responder that is administered via embedded e-mail commands. The mail-based command structure and side-effects introduced by the behavior of certain mail agents can sometimes cause problems for novice users. By its very design, Majordomo has always lacked a GUI tool to help simplify its usage for non-power users. However, with the World Wide Web providing the perfect foundation, it may be an appropriate time for just such an application.

MajorCool is a CGI application that provides Majordomo mailing list configuration and subscriber management via WWW. Unlike other Web-to-mailinglist interfaces currently available, MajorCool is uniquely different. It supports administrative as well as end-user functions, and, rather than simply generating e-mail requests, interacts directly with Majordomo files wherever possible. Developed for Majordomo 1.93, MajorCool was written in Perl 4.036 and has been successfully implemented under NCSA *httpd* 1.4 and 1.5.

Introduction

Majordomo¹ is a Perl-based mailing list manager written by Brent Chapman. Chances are, anyone who has ever subscribed to an Internet mailing list has probably used Majordomo to do it. List configuration, subscription management, identifying the members or purpose of a list, or even finding what lists are available are all functions easily provided by commands sent in e-mail to a Majordomo listserver address.

As as easy as Majordomo is to use, there are still numerous opportunities for users to encounter difficulties. Being subscribed under multiple addresses, the word-wrapping behavior of some mail agents, and even the “delayed gratification” nature of the mail-based system often confuse users. And if end-user functions such as *subscribe* and *unsubscribe* are confusing to the Internet novice, imagine the first-time list administrator staring at a complex configuration file that was requested (and must be returned) via Majordomo e-mail commands!

After the introduction of Majordomo at my site, mailing list usage by the technical community grew steadily. Having previously used only standard aliases, the engineers enjoyed the added flexibility provided by Majordomo (open lists, moderation, etc). Word about the benefits of Majordomo eventually spread to the non-technical users, and soon we had administrative assistants and product managers requesting access for purposes of maintaining customer and department lists. Here is where the trouble started.

It was soon discovered that the concept of a syntax-driven command structure was difficult for a large number of people to comprehend. Users much

preferred to interact with Majordomo person-to-person

Hi there. I am in Sally Smith's TQM organization and I would very much like to be a member of the Web-a-Holics mailing list. Thank you.

rather than be so robotic as to say

`subscribe webaholics`

Once briefed on the proper procedures required to control Majordomo, a large percentage of our Microsoft Mail using community were then stymied when they ran into the mailer's annoying tendency to wrap long text lines. Believing an exceptional job was done when they issued the

`subscribe a-really-long-list-name` command, they were frustrated in trying to understand the Majordomo error message returned, and became thoroughly perplexed when we tried to explain the concept of \-escaping long lines in order to bypass MSMail line-wrapping behavior.

These problems were exacerbated 10-fold whenever an MSMail user became administrator of a 300+ line list configuration file.

Defining The Problem

Clearly, some user-interface improvement was needed to help the non-technical users over this Majordomo usability hump, and a few members of my IS group sat down to figure out exactly what. For our PC users, a GUI tool would be a god-send. We had Microsoft *Visual Basic* and C/C++ development experience in our department, as well as a long history with Web/CGI interfaces. We were open to anything that

¹<http://www.greatcircle.com/majordomo/>

would allow us to produce a decent Majordomo graphical front-end.

We decided that we needed to support basic administrative tasks as well as end-user functions. Interface support for “subscribe” and “unsubscribe” would satisfy the majority of users, but problems in the list configuration arena were quickly becoming our number one trouble spot, so it would be desirable to address that as well.

We needed something that did not add too much extra administration overhead. Up to this point, we had managed to push most of the list administration responsibility down to the users (to the point where we never had to deal with adding new aliases whenever a list was requested – more on this later). We did not want to introduce a new tool that would needlessly insert us back into the process. Because our lists were in a constant state of flux due to a high list-owner turnover rate, we strongly desired that the GUI be in sync with Majordomo so as to minimize any per-list changes to the interface.

Taking the “in-sync” requirement even further, we felt that it was advantageous that the application interact directly with Majordomo files wherever applicable. Although we were not married to the idea of direct updates, it was felt that at least read-only access to the Majordomo support files was a viable prospect. As yet we were not sure if this would be done via NFS, server co-location, or by some other means.

Regardless of function, a final criterion established was that nothing would be made available via the GUI that was not also accessible through standard e-mail – in other words, the interface could not bypass any Majordomo restrictions or permissions.

By this time (Jan 1996) there were already a few Majordomo/Web interfaces available which we investigated, most notably Patrick Fitzgerald’s *MailServ*² and David Baker’s *LWGate*³. While both excellent tools, neither addressed the list configuration issue, our primary concern. In addition, both simply served to construct and send a syntactically correct e-mail message and did not actually provide the user with a display of the requested Majordomo function results. So, for example, any request to display current list members would be initiated on the Web but subsequently viewed via e-mail. We desired something a little more interactive. Lastly, Web interfaces to date were never assumed to be on the same system. This meant that adding a new list to Majordomo also meant updating the configuration of the Web front-end. This was just the sort of sysadmin busy-work we were trying to avoid.

²<http://iquest.com/~fitz/www/mailserv/>

³<http://www.netSPACE.org/users/dwb/lwgate.html>

Designing A Solution

We briefly considered a client/server protocol running on some TCP port, but the thought of *Visual Basic* programming (or similar) just didn’t sit right with us. With our local intranet gaining in popularity since its introduction in 1993, we felt instead that a Web-based interface would be the perfect addition to our Majordomo implementation. Our users were comfortable with HTML Form applications, and the multi-platform nature made it equally usable for both PC and UNIX systems.

The more we thought about it, the more we began to embrace the concept of CGI co-location with the mail server because it would make access to required Majordomo files a non-issue. The additional load on the server caused by the single-purpose *httpd* process was expected to be minimal.

One aspect we needed to accommodate was the fact that NCR utilizes a database-driven e-mail address resolution system called *Rolo* to support *First.Last* e-mail addressing. Our Southern California sites were additionally saddled with dual-domain support. Thus, valid addresses for our users could take on many forms:

```
Bill.Houle
Bill.Houle@SanDiegoCA.NCR.COM
Bill.Houle@ElSegundoCA.NCR.COM
bhoule@host.sandiegoca.ncr.com
bhoule
```

To enable MajorCool to understand the equivalencies between these many forms of addresses, the ability to call an external user-lookup mechanism was planned. Rather than perform a straight lexical address comparison to determine if a user was subscribed, the lookup function would return a tuple containing a regular expression pattern which would then be matched against the list addresses. This would enable the application to identify and display all subscribed lists, regardless of which equivalent address a user was subscribed under.

When it came time to design for permissions-protection of the Web interface, we decided to take the lazy approach. Rather than coding to the low-level Majordomo Perl subroutines and dealing with file-locking and other collision issues, we opted for the method utilized by past Web interfaces: e-mail! Our Web tool would interact directly with all Majordomo files for the purpose of querying status, but when it came time to modify any files, we would fall back to the usual mail-based commands. In an effort to “eliminate the middleman”, we chose to take advantage of the co-location on the server and pass the simulated mail messages directly to the Majordomo process rather than actually sending the mail.

One final design choice came after testing other Web-to-listserver tools. Most implementations provided the ability to find out details about a list before

deciding to subscribe to it. Once ready, the user would then enter an e-mail address and choose to subscribe. However, after browsing another list's details, one had to re-enter the e-mail address for subscribing. We did not like this idea of sign-in at subscription time. We chose to implement the sign-in process before giving access to list information. Not only did this match Majordomo's authentication requirements (where many list query functions are unavailable depending upon the requester address), but it also saved the duplicate entry of subscriber address in a case where multiple lists were selected within a single session.

Implementing A Solution

Development of MajorCool began by focusing on the user-interface issues of the *subscribe* and *unsubscribe* functions. The first task was to create the interface to the user-lookup program in order to determine subscription status for all possible addresses. By utilizing our *Rolo* database, not only were we able to create a regular expression to match potential subscribed address, but we were able to treat the function as a name-to-address mapping routine. We designed the process to accept a fullname rather than e-mail address and convert that to a list of e-mail patterns. Since *Rolo* supported the concept of nicknames (Rob=Bob=Robert), we were able to generate a regexp of all possible *First.Last* name/nickname permutations as well as the standard *user@host* addresses.

To provide some modularity and allow other sites to implement a different user-lookup function without impacting the rest of the MajorCool code, the user-lookup function was placed in the MajorCool configuration file as a generic *siteaddr()* function and also included a *%siteaddr* string array to define the HTML prompts, as well as a *by_siteaddr()* function to allow a sort mechanism to be associated with the address types. A sort mechanism allows lists to be displayed, for example, in domain order to make locating a user easier.

Implementing subscribed and unsubscribed lists as HTML checkboxes was straightforward. The difficulties began when one considers the fact that CGI scripts are stateless applications, and we needed to pass the submitted list of checkboxes back to MajorCool for the second phase of subscription processing (and eventual hand-off to Majordomo). We did not want to waste time unsubscribing to previously unsubscribed lists, nor did we wish to re-subscribe to ones already in effect. Therefore, we built a "before and after" state picture of subscription choices and submitted only the differences.

A problem with subscriptions remained: how to deal with Majordomo's concept of "unadvertised" lists. With unadvertised lists, users may be subscribed to lists that they cannot even see on the form, yet we still needed to pass their status to the next phase. To handle this, we used HTML "hidden" fields to

maintain the state of unadvertised lists when building the list of checkboxes. This enabled us to keep an accurate view of membership status as the form was submitted to phase two of the subscription processing.

The next obstacle was slow performance. The subscription management screen of Majordomo provides basic list info such as the list name, subscribe policy (closed/open/auto), and list description. All of this information was obtained through the *get_config()* function call provided in the Majordomo libraries. We soon found that invoking *get_config* on any multiple of lists became unbearably slow. While more than adequate for offline mail-based processing, *get_config* was death to an interactive application.

To eliminate the performance problems caused by *get_config*, we simply stopped using it. We developed a "key cache" file that contained all of the information required for a basic subscriber view of each list. Although the cache is a simple flat-file database, it is able to pre-load much faster than multiple calls to *get_config* ever would. The only problem introduced by this design change was the added maintenance of the cache file, but this is easily managed via a cron job set at some reasonable update interval.

In retrospect, the subscription management portion of the user-interface was the most troublesome to implement given *Rolo*, the unadvertised lists, "key cache" workaround, and state maintenance between phases. The remainder of the interface, from password-protected list administration to HTML *textarea* fields for *list.info* and subscriber file management proved relatively easy in comparison.

The next hurdle was converting the user interface selections into Majordomo-style mail commands. Subscription management changes were already supported, but we were still left with the task of modifying the list configuration file. Since only a subset of configuration fields are supported by the interface, we were unable to build and submit a complete configuration file out of the supported fields. This left us little choice but to perform an in-stream modification of the existing config file through string substitutions. Although effective, this limited our support to *variable=value* configuration items only.

Because MajorCool requires read access to all the Majordomo per-list files, the tool needed to operate with the appropriate permissions. The easiest way to accomplish this was through an intermediate shell script "shim" which invokes the real MajorCool program via the *suid* Majordomo *wrapper*. Thus, MajorCool operates with the same privileges as Majordomo itself.

The use of the *wrapper* program solved one problem, but introduced another. Access was needed to CGI environment variables, yet the *wrapper* is specifically designed to provide a minimum environment space for security reasons. Therefore, in order to support these CGI variables, an additional function of

the CGI "shim" is to convert any needed *httpd* environment variables to command-line arguments that the *wrapper*'ed MajorCool could use.

Configuring MajorCool

From an end-user perspective, the *cgi-bin* "shim" is the heart of MajorCool and therefore should be named appropriately for usage as a URL – for obvious reasons we use *majordomo*. But from a system standpoint, the invoked main script is the true MajorCool workhorse. This script parses its command line for any needed arguments, then reads */etc/majordomo.cf* and its own configuration file to obtain the basic details about the Majordomo

configuration and how the Web interface should operate.

The *majorcool.cf* configuration file defines the basic appearance and behavior of the MajorCool application. Site name, the use of pick-list or type-in field for list administration selection, and *siteaddr*-specific customizations are defined in this file. A sample configuration is included below.

```
$site = "NCR Southern California";
$siteinfo = "www/majorcool.info";
$header = "Page.Header";
$footer = "Prog.Footer";
$hidden_admin = 0;
$hidden_admin_name = "MajorCool";
$hidden_lists = 0;
$hidden_passwd = 1;
```

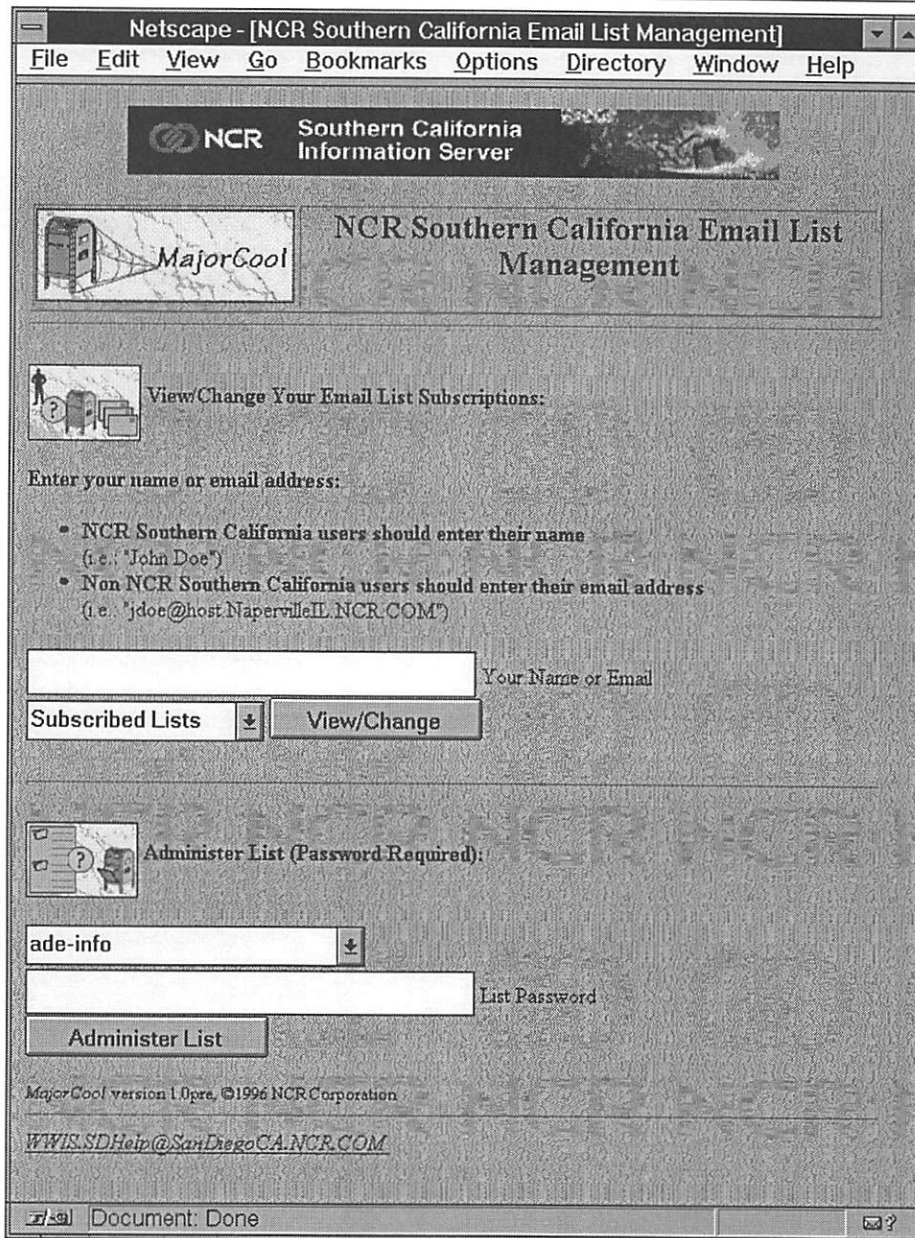


Figure 1: MajorCool Opening Screen

```

$log_suffix = "last";
$member_advertised = 0;

#####
# This "siteaddr" LOOKUP section defines local
# address customizations and prompts.
#####

%siteaddr = (
    'prompt',    "Your Name or Email",
    'intro',     "<B>Enter your name or email address:</B>"
    [...omitted....]
);

sub siteaddr {
    local($target) = @_;
    #
    # NCR employees may be subscribed to lists as:
    #   First.Last
    #   First.Last@Domain
    #   login@Domain
    #   login@host.Domain
    #

```

```

# This implementation example uses a
# 'ROLO' tool to map
[...omitted....]
return ($user, $preferred_addr, $regexp);
}

sub by_siteaddr {
    [...omitted....]
    $x cmp $y;
}

```

Using MajorCool

MajorCool is invoked through the `cgi-bin` URL. It can be configured to run separately as end-user and list-administrator functions, or combined in a single screen interface. The screen capture shown in Figure 1 is the unified version.

Netscape - [NCR Southern California Email List Management for Bill Houle]

File Edit View Go Bookmarks Options Directory Window Help

NCR Southern California Information Server

NCR Southern California Email List Management for Bill Houle

Shown below are your subscribed email lists:

- Use check-boxes to subscribe/unsubscribe to lists.
- Click on the list name to view list details.
- indicates the list is locked or closed. Your request will be sent to the list owner for approval.
- indicates that the list is unconfigured and therefore cannot be subscribed to at this time.
- Your actions are logged. Please do not maliciously change the subscriptions of others.

Apply Changes Reset Form

	List Name	Description
<input checked="" type="checkbox"/>	corp-pubs-insight	Corporate Publications -- "Insight" and Others
<input checked="" type="checkbox"/>	ms-newsletter	Microsoft Newsletters
<input checked="" type="checkbox"/>	ms-nt-newsflash	Microsoft Premier Support NT Newsflash
<input checked="" type="checkbox"/>	ppp-users-sd	
<input checked="" type="checkbox"/>	virtual-office-coordinators	So Cal Virtual Office Coordinator Group
<input checked="" type="checkbox"/>	webaholics	WWW Therapy Group
<input checked="" type="checkbox"/>	www-authors	Infobahn Docmasters and HTML/WWW Authors
<input checked="" type="checkbox"/>	wwwteam	Southern California World-Wide Web Team

Apply Changes Reset Form

MajorCool version 1.0pre, ©1996 NCR Corporation

WWIS.SDHelp@SanDiegoCA.NCR.COM

Document: Done

Figure 2: Checkbox subscription display

All displays incorporate a configurable header and footer to allow a site to maintain a common look-and-feel between the MajorCool application and the rest of the intranet.

After establishing an e-mail address, the end-user screen displays all subscribed lists (or alternately, all available lists) in the form of checkboxes; see Figure 2. Users may toggle between subscribe and unsubscribe status by changing the checkbox selections, and upon submission, the appropriate commands are issued directly to Majordomo (bypassing *sendmail*).

The description of the lists are shown in the display, with further details available by clicking on the hyperlink of the list name. Traveling to the list details screen will allow access to information such as the *list.info* file, number of subscribers, subscriber list (if permitted) and more; see Figure 3.

Access to the list administrator functions are protected by the standard Majordomo per-list passwords. Once the password has been accepted, the list administrator is presented with a screen of some of the more commonly-modified aspects of Majordomo *list.config* file management; see Figure 4. As before, additional details are available by selecting the active hyperlinks. Through these links, one may also view or modify the *list.info* file, as well as see/alter the subscriber list itself.

Just as with end-user subscription changes, when the modifications to the list configuration are submitted, the changes are converted to standard Majordomo command syntax and handed to Majordomo without using e-mail as the intermediary. Unlike the subscription changes (which operate with the name & address of the end-user), list-configuration changes may be submitted with a configurable address. This is useful when multiple administrators manage the list and actions need to be submitted as other than the default list owner.

You can see that a pattern has developed: all "view" operations are supported directly through the MajorCool interface. However, any "change" operations are converted into equivalent Majordomo commands and handed to the Majordomo/*wrapper* duo for processing. Upon successful completion, the user will receive normal mail confirmation just as if the commands were issued via e-mail. As much as it might have been nice to edit files directly rather than submitting to Majordomo, this method certainly proved safer and infinitely quicker to implement.

Experiences with Majordomo

I believe the Internet community owes a great debt of gratitude to Brent Chapman and the entire Majordomo-workers community. The Majordomo product is well-supported, easily extensible, and most

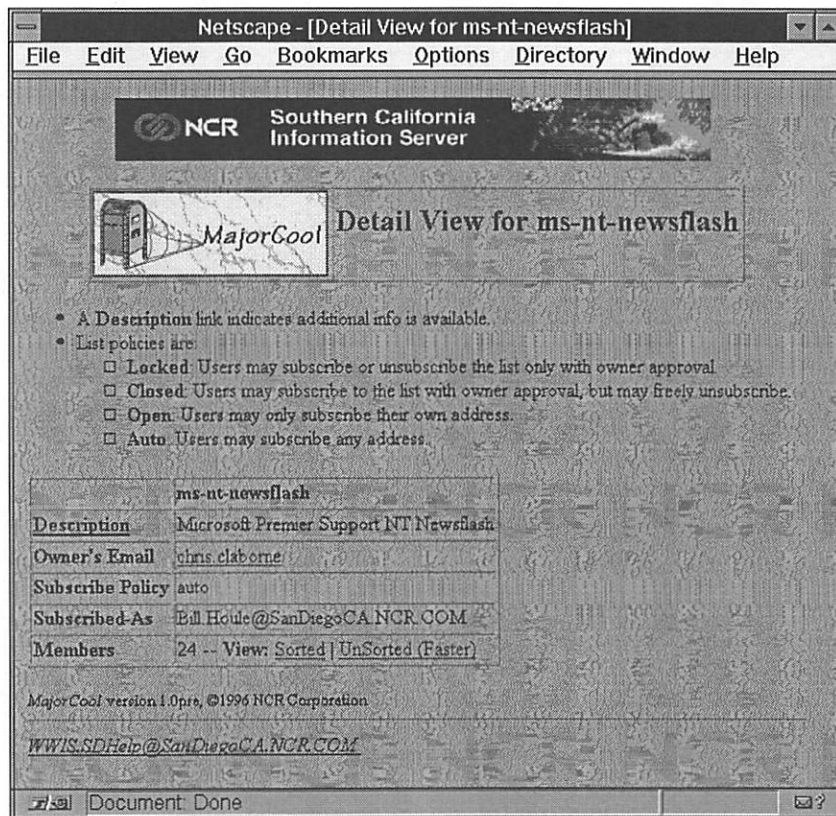


Figure 3: List details

importantly, widely recognized and accepted by end-users. Brent's initial work was featured at the *LISA 6*[1] conference.

As evidence of Majordomo's extensibility, the fact that MajorCool was not implemented on a "plain vanilla" Majordomo installation needs to be explained.

Over the course of Majordomo implementation at our site, we have made numerous minor enhancements to the default distribution in order to meet the requirements of our users. Such minor enhancements include the addition of new keywords in the list configuration files that would, for example, control whether or not list names are included in the MSMail Global Address List or whether the sending of the

"Welcome" message to new subscribers was On or Off.

Support for these options are included in the MajorCool release, but are not required and can easily be removed. However, there are two additions that are important to the operation of MajorCool, and a slight amount of re-work would be required to make it operate on a default Majordomo distribution.

The first major change is the use of an *owner* keyword in the configuration file. As I mentioned earlier, our lists have a high degree of entropy due to changing job functions and department roles. We found that list owners changed relatively frequently, and we were not thrilled with the idea of constantly tweaking aliases whenever someone changed jobs.

Netscape - [Configuration Options for list: ms-nt-newsflash]

File Edit View Go Bookmarks Options Directory Window Help

NCR Southern California Information Server

MajorCool

Configuration Options for list: ms-nt-newsflash

- Reload this page after submitting changes to view the effects of your changes. (In some browsers you will need to go up a page and come back.)
- The Info File is sent to all new subscribers, or by request via the *info* command.
- Change the *apply-as* field to have Majordomo results sent to other than the list owner.
- Additional help is available for all keywords.

Description	Microsoft Premier Support NT New
Owner's Email	chris.claborne
Directory Export	<input type="radio"/> yes <input checked="" type="radio"/> no
Info File	Last modified: Fri Jan 19 12:45:04 1996
Subscribe Policy	<input type="radio"/> locked <input type="radio"/> closed <input type="radio"/> open <input checked="" type="radio"/> auto
Administrivia	<input checked="" type="radio"/> yes <input type="radio"/> no
Admin Announcements	<input checked="" type="radio"/> yes <input type="radio"/> no
Send Welcome Msg	<input checked="" type="radio"/> yes <input type="radio"/> no
Max Message Length	0
Moderated	<input type="radio"/> yes <input checked="" type="radio"/> no
Moderator	
Approve Password	ms-nt-newsflash.pass
Hidden Membership	<input type="radio"/> yes <input checked="" type="radio"/> no
Members	24 -- View/Edit: Sorted UnSorted (Faster)
Last Changed	Sun Jun 30 22:15:48 1996

Apply Changes as chris.claborne

Reset Form

Document: Done

Figure 4: List configuration screen

Our best defense was to push that ownership responsibility back down to the users. By implementing an *owner* keyword (which is a reference to a valid e-mail address), list admins could re-assign list ownership to another simply by modifying their configuration file. A cron job on the system regularly parses all config files to build a *sendmail* alias file from the entries. This additional feature makes our sysadmin life easier because it removes us completely from the business of changing list owners. Everything is automated and hands-off. Although the enhancement is not critical to MajorCool itself, there is nonetheless liberal use of `$config_opts{'owner'}` in the code that would have to be changed if this keyword were not present.

The other Majordomo modification was required to support the full-text editing of the subscriber list on the list administration screen. It is actually another case of laziness rearing its ugly head. We did not feel motivated enough to implement a *diff(1)* routine within MajorCool to process changes to the subscriber list, so we instead implemented a *newwho* Majordomo command to provide wholesale replacement of the list. The command, similar to *newinfo* (with almost identical code), simply replaces the current list with the input provided. The in-line list editing function will not work without this added function. A side-effect is that list members are not greeted with the usual Welcome message when added using *newwho*. For this reason, an *approve* sub/unsub mechanism is also provided on the admin form for times when adding individual users is preferred.

Experiences with MajorCool

The bulk of MajorCool took approximately one month of part-time work to implement. Naturally, it has undergone a steady stream of revisions as bugs were fixed and new features were added. A minor rewrite phase required an additional two weeks as the code was made more configurable and less NCR-specific in preparation for release back to the Majordomo community. MajorCool is now approaching 1600 lines of code.

During beta-testing of the product, a list administrator remarked that the tool was "majorly cool". Given that we were still seeking to name the application, MajorCool seemed an appropriate choice.

Majordomo/MajorCool in NCR is currently supporting 105 mailing lists, the largest of which contains over 2000 subscribers. We support an average of 650 "change" requests per month, 30% of which are made via MajorCool. A significant data point is the fact that, using the number of requests-per-month as an indicator, overall Majordomo awareness has increased by 250% since the announcement of MajorCool availability in March. MajorCool, in essence, seemed to help promote Majordomo.

User reaction to the MajorCool tool has been marked by a distinct lack of "help me" mail sent to

the *majordomo-owner* address; for this reason alone I consider MajorCool a success. Since replacing the *majordomo-owner* alias with a canned response referring people to the MajorCool URL, e-mail to the alternate "human" address has dwindled down to nothing.

MajorCool as a GUI tool extends the functionality of Majordomo. Most notably, its unique features address two of the problem areas previously identified by Brent in his original *LISA* paper. The externalized user-lookup function (the NCR *Rolo*, in this case) corrects the problem of identifying duplicate address formats while the Web-based nature of the interface eliminates line-wrapping problems caused by other (mostly PC) mailers. Finally, the administration capabilities help simplify the process of list management.

Future Work

As of this writing, we are on the cusp of a Majordomo 1.94 final release, and MajorCool has not been tested against the new version. First and foremost in the way of future work would be this undertaking.

Another much needed addition is the support for string array variables in the list configuration mode. You'll recall that a new configuration file is submitted by performing an in-stream string substitution on the var/value pairs. The multi-line string array data type does not fit this regexp pattern which is unfortunate because support for such array variables as *message_fronter* and *advertise/noadvertise* are sorely missed.

Archive support would also be a welcome enhancement to the MajorCool command suite. Provision for the Majordomo *get* command immediately comes to mind, but the idea of list archive management can be taken even further if support for Web-based archive browsers such as Kevin Hughes' *Hypermail* were added.

Acknowledgements

I would like to acknowledge the contribution of Matt Bateman, who was involved in the initial design and coding of MajorCool prior to his leaving NCR.

Availability

MajorCool was developed at the NCR Corporation with the full intent to release it back to the Internet community. For my own reasons of convenience, it is available for download via World Wide Web access at <http://www.DigitalMasters.COM/~bhoule/> rather than from the NCR external site. It is freely modifiable and distributable provided all copyrights are maintained.

It is hoped that other Majordomo users will benefit from the MajorCool release and install it at their sites. From this, we can build upon the base of the *Majordomo-workers* and *Majordomo-*

users support groups. I will be monitoring these lists in support of the MajorCool application, and I encourage others to enhance the product further to make it truly useful for everyone.

Author Information

Following his Math/Computer Science & Cognitive Science undergraduate work at the *University of California – San Diego*, Bill Houle has spent eight years at the NCR Corporation. Working variously in Software Engineering and support, he has spent the last five years in the IS area, currently as the *Information Access* Team Leader. In his present position, he is responsible for Internet services (E-Mail, News, World Wide Web, etc) at the two Engineering facilities in Southern California. He can be reached via U. S. Mail at NCR Corporation; 17095 Via Del Campo San Diego, CA 92127. Reach him electronically at Bill.Houle@SanDiegoCA.NCR.COM.

References

- [1] Chapman, D. Brent, "Majordomo: How I Manage 17 Mailing Lists Without Answering 'request-' Mail", *Proceedings of the Sixth Systems Administration Conference (LISA VI)*, 1992, pp 135-143.

The PGP Moose – Implementation and Experience

Greg Rose – Qualcomm Australia

ABSTRACT

USENET news traffic is growing exponentially, at a rate only barely less than the World Wide Web. Many observers believe that moderated newsgroups are the way of the future, to keep content high quality and avoid advertising. However there is essentially no security enforced on moderated newsgroups, and only a very small chance that decent security could be integrated into the framework itself.

The PGP Moose is free software, which attempts to address this situation. The aim of this software is to monitor the news postings of moderators of Usenet newsgroups, and to automatically cancel forged messages purporting to be approved. This software and protocol is designed around cryptographic signatures. The protocol is designed to allow the use of different signature techniques. The current implementation assumes the use of PGP [1] (Pretty Good Privacy) signatures, but can be easily modified to use others, such as the Digital Signature Standard or MD5. PGP was chosen for its widespread availability around the world.

Since first being made available a few months ago, the PGP Moose has been placed in control of a number of newsgroups in the USA, Germany and Australia. It is also used by a few individuals, particularly in Germany. Other solutions to the same problem are also in limited use.

In this paper I examine the history and implementation of the PGP Moose, and some of the problems with its deployment and debugging.

How Does It Work?

This document is written from the point of view of a newsgroup moderator, but individual users could also use the facility provided by the PGP Moose in analagous ways.

Usenet News works simply by flooding the network. Cooperating news server machines negotiate to decide which articles they have not yet received, and then get them before passing them on in turn. This protocol is called *NNTP*, Network News Transport Protocol [2]. This is surprisingly efficient in use of bandwidth, which is why there is so much news being written. It is also a wonderful way to distribute advertising material, an act called *spamming* after the menu offered in a Monty Python sketch.

Moderated newsgroups function differently. The news system control files mark particular newsgroups as being *moderated* and when the normal mechanisms attempt to post news to such a group, the message is

intercepted and instead is mailed to the moderator. The moderator marks the message as approved and resubmits it, and this time the news software recognises this and enters it into the system.

The problem is that there is nothing, really, preventing any old user marking the message in the same way. The special mark is simply a header line *Approved:*, usually with the moderator's name and email address on it. As far as the *NNTP* software is concerned, though, the existence of the header line is a seal of approval.

When a moderator wants to protect their group from forged/unapproved postings, they should register their interest with one or more of the sites running PGP Moose, and pick up the submission script. As part of this process, the moderator would specify one or more PGP public keys that are allowed to approve postings.

When a post comes in, and the moderator wishes to approve it, they do whatever they would normally

```
X-Auth: PGPmoose V1.0 PGP sci.crypt.research
iQBVAguBL1/Kg2zWcw3p062JAQEYIgH/Xyrz6LaGG+fHaSxoexMECovzkIoADrQx
173IX1UQEIoFl5jnDBBdHVvqTMEPS0118ytYVQZoQrdStuXB9Oc9gQ==
=azqs
```

Figure 1: PGP header

do before actually using inews (or whatever) to post the message. As their last action, they run the PGP Moose Approval program “pmapp”. This inserts another special header which looks Figure 1. If there are multiple moderated newsgroups, there might be multiple X-Auth: headers, one for each group that has requested assistance from the PGP Moose daemon. In this example you can see that the authentication carries the name of the authenticating program, a protocol version number, an identifier of the type of digital signature (currently only PGP) and the name of the newsgroup in question. These, as well as the From:, Subject: and Message-Id: lines, the list of newsgroups, and the non-blank lines of the message itself, are used as input to the PGP program to generate a signature.

The lines of the message body are preprocessed in a way that is meant to render harmless any mangling that a typical news system might do to the article. The article itself is not changed, only the input to the signature generation. If a news system subsequently mangles the article in a “normal” way, for instance by inserting a “>” in front of a line starting with “From”, it will still pass the signature check.

The list of newsgroups must be handled specially, so that an article posted to multiple moderated controlled newsgroups can be appropriately handled. See below for a more detailed treatment of the issues of posting to multiple moderated newsgroups.

The PGP signature is then inserted into the X-Auth: header, mostly so that it won’t interfere with, or be confused with, any signature in the body of the message.

Anybody can check whether the message has been modified in any significant way, simply by running the PGP Moose Approval Checking program “pmcheck”. More importantly, though, the sites running the PGP Moose Checking Daemon will be doing this automatically for every posting to the registered newsgroups, or from the registered users. And, if a posting fails the checks, it may be automatically cancelled, and a notification sent to the moderator.

The important things to notice about this process are that the actual news transport software isn’t changed in any way, and that it provides a sort of “morning after” protection. A knowledgeable person

can *still* send spams to moderated newsgroups, but they will be automatically cancelled soon thereafter.

The Bits

The approval and checking part of the PGP Moose consists of a number of Bourne Shell scripts calling standard Unix utilities and PGP. I could have used perl more elegantly, but this stuff is marginally more widely available. If there are Unix version dependencies, they should be considered to be bugs and I’ll happily attempt to remove them.

pmapp usage: pmapp [newsgroup|user] [file]

This script takes the not-yet-posted article, specified either by filename or from standard input, and creates a signature for it, which is then inserted in the X-Auth: header. The article, ready for posting, appears on the standard output.

In the configuration section at the top of the script, the moderator may build in the default name of the newsgroup or user, PGP User Id to be used for the signature, and the corresponding password. This is simply for convenience, since spammers are not so likely to go cracking the computer to get the password, and it is a relatively simple matter to generate a new user if it is, indeed, compromised. For the paranoid, like myself, if the password is not configured into the script it is read from the terminal instead.

pmcheck usage: pmcheck [newsgroup|user] [article]

This script takes the article, specified either by filename or from standard input, and checks that the X-Auth: line is something it considers to be correct and that the article has not been tampered with. Pmcheck returns successfully if everything checks out. Otherwise it will return failure and issue one of a number of error messages, see Figure 2.

Anybody can run pmcheck. It behaves slightly differently depending on the existence of a file called (by default) PGP_Moose_accept, and the presence or absence of a newsgroup or user argument. This file, if it exists, should contain lines with a newsgroup name or email address, some whitespace, and the PGP User Id

```
Posting for $NEWSGROUP not approved with PGP Moose.
Invalid designated signature from $GROUP
No public key for signature $GROUP
Signature doesn't match $FILE for $GROUP
```

Figure 2: Pmcheck error message

```
sci.crypt.research      moderator <ggr@sydney.sterling.com>
sci.crypt.research      moderator <pgut01@cs.auckland.ac.nz>
ggr@sydney.sterling.com Greg's News <ggr@sydney.sterling.com>
```

Figure 3: Multiple moderators

approved by the moderator or user (usually made up specifically for this purpose). Multiple lines for the same newsgroup/user are allowed. For example, see Figure 3. If such a file exists, and a specific newsgroup or user is specified, pmcheck is silent if all is well, and issues the last of the error messages above if everything else was all right but the signature was from the wrong person. There must, in this case, be a signature applying to the designated newsgroup or user.

Without such a file, or if no specific newsgroup or user is given, all the signatures in the article are checked. In this case it is not considered an error if the signature cannot be checked due to a missing public key. If each signature is otherwise valid you will get a message like:

Valid signature from '\$SIG'.

In any case, if there is a problem with a signature mentioned in the PGP_Moose_accept file, it will be reported and an error status will be returned.

pmcanon

pmnewsgroups

These two scripts are used by pmapp and pmcheck to recreate the exact input for the signature, and to extract the list of newsgroups in the header, respectively. More documentation is in their manual pages.

The PGP Moose checking daemon is packaged separately, as there would not seem to be a lot of value in having too many people running it. Accordingly, I was less precise in making it run absolutely everywhere. It requires the Korn shell or equivalent, and perl, and currently only interfaces to INN. I expect it would be easy to interface it to CNews, but I don't have one.

pmdaemon

Runs pmcheck to check the X-Auth: header for each controlled newsgroup for each article that arrives in an appropriate newsgroup. Mail is sent about any errant articles, and automatic cancellation may be enabled.

pmcancel

prepares a cancellation message based on the headers of another message.

When (if) I get a chance, I will create mail server scripts that allow moderators who are not using Unix to use these facilities. The first allows a moderator to mail a PGP signed copy of the article to be posted. The server will then verify that the moderator sent it, and post it with a (different but corresponding) approval. The second will accept an article and return something that you can check the signature on. Either way, any moderator will still need PGP.

How Do You Register For The Service?

Ahhh, this is the hard part. After all, it would be pretty undesirable if someone, meaning well, took any old body's word for it that some important moderated group should start working this way, before the moderator was able to start approving postings. A great way to hijack a newsgroup. Similarly for hijacking some other user's postings (tempting though it might be :-).

Another possibility is that someone, having seen what the valid signature looks like, simply creates a whole new PGP key that happens to have the same PGP User ID. Then they can sign and post stuff too.

The solution to both of these problems is the classical one for public key systems. You need either a certifying authority or the PGP Web of Trust. We're using the Web of Trust. This means that the moderator of the newsgroup must establish a path through only other people who are trusted, to the administrator of one of the PGP Moose checking Daemon sites.

For each newsgroup that wants to utilise this program, the moderator will have to create a special PGP key pair (preferably 512 bits to keep the X-Auth: lines short), and sign it. They must then establish a path of trust to someone who is running the PGP Moose server. It will be up to the administrator of that server to make sure that only trusted moderators' keys ever get into the server's keyring. In one case I achieved this by telephoning the University switchboard and asking for the moderator, then verifying the key's signature over the phone.

There can be no shortcuts to this procedure. Otherwise we are all back where we started.

In the case of an individual user, again you should establish this verification path to one of the administrators of the PGP Moose service. Contact me <ggr@usenix.org> for the time being to mutually figure out how to do this.

Handling Multiple Moderated Groups:

When I first proposed this tool, I was under the impression that posting to multiple moderated groups was an aberration that should be stamped out. This turns out not to be the case, and revisions to support this have been the cause of some delay in the deployment of this tool.

When the news system sees that an article has been posted to one or more moderated groups, it checks for an Approved: header. If the header exists, the article is accepted and processed normally, otherwise it is mailed to the moderator of the first moderated newsgroup mentioned in the Newsgroups: header. There seem to be three cases of interest.

The trivial case, and the most normal one, is that there is only one moderated newsgroup mentioned. The moderator approves the posting, and it is done.

The next, and probably most important, case, is when a moderator wants to cross-post a FAQ to their own group, as well as news.answers (for example). In this case their approval counts for both groups, so they can insert the Approved: header and post away. Presumably the other groups are not under the control of the PGP Moose Daemon. In this case the moderator can just go ahead and put in the Approved: header, and save themselves and pmapp a lot of time. It will be passed right through.

The other case is harder to get right. This is when the article really is meant to be posted to two (or more) unrelated moderated newsgroups. Currently, if the first moderated group's moderator approves the posting, the other ones never hear about the article, at all. If this second group is controlled by the PGP Moose an automatic cancel will be generated. So it becomes very important for the moderators to do what they should have been doing already, namely forward the article to the next moderator. This tool can't help people who don't use it, but it provides some support for those who do.

The approval script checks whether there are any moderated newsgroups left that don't have X-Auth: headers for them. If there are none left, an Approved: header is inserted and the article gets posted. Otherwise, it issues a warning, and re-orders the newsgroups header with a newsgroup which is moderated but has no X-Auth: line at the start. When the article is posted, the news system will forward it to the moderator of the (new) first moderated group. If all moderators are sensible, and check for moderated newsgroups in this fashion, the mess should sort itself out and the last moderator will go ahead and post it. A warning message to the subsequent moderator NOT to change the article is also inserted, since such a modification would invalidate the previous signatures.

To ease this process, a second type of X-Auth: header is supported. This has the form:

X-Auth: None ... Newsgroup

The important fact about this is that the newsgroup appears last on the line, allowing a sort of partial approval, from moderators who don't use the PGP Moose.

The Newsgroups: line is split into a sorted list of newsgroups in order to generate the digital signature. Note that this means that once an article has been approved and authenticated by one moderator, it cannot be altered in any way by a moderator of a subsequent group, including altering the set of newsgroups mentioned in the Newsgroups: header, the body of the posting, or the other headers mentioned above.

Possible Problems

If an article is truly mangled, e.g., by truncation, it will fail the authentication and be cancelled. Until it is demonstrated otherwise, this is assumed to be a rare

and minor problem. When a cancel is issued, mail is sent to the moderator of the group telling them, and they can tell us if it becomes a problem.

Currently the signature produced is assumed to be a PGP version 2.6 compatible one.

As with most things, the true vulnerabilities don't necessarily show up straight away. The ones that I have thought of are pretty simple. (I'm thinking in terms of its primary purpose here, for moderators to sign approved postings, and auto-cancellation of anything not properly approved when it reaches another site.)

If you have access to modify news as it is being propagated, and you wanted to harass a poster or the moderator, you could intentionally alter the piece of news. Without the PGP Moose, your alteration would go undetected, so you could change the sentence "I love Newt" to "I hate Newt" (or vice versa) and damage the poster's reputation (and maybe get them sued for slander). With the PGP Moose, the change will be obvious and the article will be auto-cancelled. The attack here is that it is a very anonymous way to get articles cancelled – the PGP moose does it for you, and there is no way to trace down that you caused the corruption.

Suppose someone posts something badly thought out in a moderated group. You could keep a copy of the posting, and keep reposting it every few days. This would make the original author and the moderator of the group pretty upset. Note that the date of the approval is part of the signature, so there is no question about whether or not the originator really reposted it himself, but it would still be annoying.

The PGP Moose only includes certain headers in its signature. You could take an article and insert other headers (anything starting with "X-" is otherwise ignored) which offend people. However the really important ones can't be fiddled.

Status

These scripts are implemented already, except as noted above. The approval scripts are considered production quality, and can be obtained from my web page. The checking daemon is also in day-to-day use and is available via the World Wide Web interface mentioned. There is a mailing list (maintained using Majordomo) called pgp-moose@ornl.gov for people using and maintaining the software.

In the meantime, if you want to use the tools, or particularly if you want to run a PGP Moose checking daemon, contact me (ggr@usenix.org).

Obtaining, Installing, Configuring

I regret that I don't have a public ftp site, but I do have a web page where you can get a shar archive of the approval code. It is <http://usenix.org/~ggr/PGP-Moose.shar> off my home page.

It is hard to talk in detail about installation and configuration, since many users are not in charge of their own news server configuration. In my case, I run all of the things out of a subdirectory of my home directory. The only thing outside this area which must be changed is the INN newsfeeds file, if you are running the checking daemon. So, get the distribution file as above and unpack it wherever you want it to live.

There are configuration sections at the top of pmcheck, pmapp and pmdaemon. I like to think that they are relatively self-explanatory. One of the harder decisions is whether to use a separate keyring for PGP Moose applications or not. It is very strongly recommended that you do, if you are going to run the PGP Moose checking daemon, as the keyring files will need to be readable by the userid which INN runs under (usually "news"). Most of these options can also be overridden by environment variables or command arguments, so it is possible to leave the scripts unmodified and simply put a wrapper around them (which is what I do).

In the case of pmapp, the newsgroup or user that the authentication applies to can be specified on the command itself; The PGP user id and password, and the Approved: header's contents, can be specified by environment variables PMUSER, PMPASSWORD and APP, respectively.

For pmcheck, the important one is the name of the configuration file specifying which signatures are valid for which newsgroups or users.

Pmdaemon runs from INN, and needs some special care to set it up. "news" needs access permission to the directory and files for PGP Moose, and also read permission on the public keyring. Note that PGP creates keyrings with only owner permissions. The search path is rarely correct, and should be set at the top of the pmdaemon script. There are also a number of file names and mail addresses, but the comments should be clear enough.

Lastly, you want to incorporate pmapp in your moderation script and possibly your posting script. In my case, the last line of my posting script basically said

```
/usr/local/news/inews -S -h <tempfile
but now it says
pmapp <tempfile | \
/usr/local/news/inews -S -h
```

To authenticate postings as an individual (as opposed to a moderator) I had to take a copy of the installed Pnews script, make sure it came earlier on my search path than the normal one, and modify that. You have to be careful that no extra signature files get appended after the pmapp is executed. Again, immediately before the "inews" call is the right place. I'm not sure whether this will work for all versions of news, this is not really my field of competence.

History and Experience

A few years ago, the idea of starting to post advertising to Usenet in bulk quantities started to take hold. The law firm Canter & Siegal raised this to an art form and even published a book about how to do this. In retaliation, a person who chose to call himself "Cancelmoose" started to forge cancellation messages for such postings. At the time, the very thought of forging cancel messages was considered a sin. After a while Cancelmoose (the person) created Cancelmoose (the program) to do the job automatically.

In early 1995, and again in April, there were a couple of spams which carried Approved: headers. I had recently become a co-moderator of sci.crypt.research and was at first amused, and later depressed, by the proposed solutions flying back and forth on the Moderators' mailing list. (Some moderators asked to be removed from the list due to the noise level!) I wrote the first version of the PGP Moose scripts, but without any news system hooks and without the capability to handle multiple moderated newsgroups. Things went quiet for a while, and I was waiting for a collaborator who knew more about NNTP and INN than I did to integrate it with news. In the meantime there was a reasonable amount of feedback regarding the need for support of multiple newsgroups in particular.

In September I gave up on my hypothetical collaborator and learned enough about INN to integrate the PGP Moose into posting and checking roles. I had spent a lot of time on the multiple moderators problem. I re-announced the software and got a number of interested parties, but it was now six months since the last bulk spamming of moderated newsgroups.

The PGP Moose fully controls postings to comp.society.privacy, soc.russia.moderated, misc.news.bosnia and comp.std.c++, as well as a number of German newsgroups which I don't see, and partially controls sci.crypt.research. By "partially controls", I mean that not all of the moderators use the tools, so the checking daemon verifies that, in the absence of a strong digital signature, at least the "approved" line is one that it is prepared to accept. Four or five individuals also use it to approve their personal postings. It is also incorporated in Igor Chudov's robo-moderation software.

Debugging the software has proven interesting. DEC's ULTRIX has an amazingly primitive shell as its /bin/sh, which took a week or two to track down.

The problem with a digital signature of any kind is that you get no indication of what went wrong. In the very rare cases when a digital signature failed to check out when an article arrived at my machine in Sydney, we had to attempt to recreate the input correctly for comparison. Since the moderator then sent

that input to me by e-mail, there was a finite possibility that the same mangling happened along the way. PGP encryption and ASCII armour has proven invaluable for getting the *exact* input from one place to another. One bug involved trailing spaces on Newsgroup: or Subject: lines.

Not all of the problems which need to be handled come from the PGP Moose itself. The posting software used by one of the `comp.std.c++` moderators insists that there should be a space after every colon on the subject line, but it puts it there after the signature is generated. This is interesting given the use of “:” in C++.

Except for things which could be called deficiencies in the design, there has not yet been any identifiable corruption of an article in transport. Thus one of the bugbears of the original discussion has simply failed to materialise.

Len Levine, moderator of `comp.society.privacy`, has been a great help with testing the software. He posted two copies of a message describing what the PGP Moose was intended to do, one with authentication, and one without, and asked for replies from anyone who saw only one of the two messages. Out of an estimated readership of 10,000, there were two replies, and one of the people had replied to the message which did have the extra header! He also intentionally mangled one article to verify that the software noticed, which it did.

The big challenge at the moment is to get more sites to run the checking daemon, and to use the power of the digital signatures to automatically distribute configuration information. Currently `ornl.gov` and `qualcomm.com` are running well-connected Checking Daemons.

Other Efforts

The newsgroup `comp.os.linux.announce` uses PGP to sign the body of each valid message. Its moderator, Lars Wirzenius, is more concerned that his readers be able to verify the postings for themselves than with automatic cancellation.

The moderator of `comp.dcom.telecom` uses a very simple scheme with MD5 hashing of the body of the message with a secret password. He has trusted users running a copy of the script at other places to generate automatic cancellations. The only problems with his approach are the issues of sharing the secret key around, which prevents this solution from scaling well, and the fact that the headers of the message are unprotected.

David Lawrence, more commonly known as `tale@uunet.net`, has begun using a similar concept for validating newsgroup control messages. In his case there are patches to be incorporated into the news servers. It is a pity that these two solutions to similar problems are not better integrated.

Author Information

University of New South Wales B.Sc. Worked for Fawnray Pty Ltd (became Neology Ltd), Softway Pty Ltd, Visiting Scientist at IBM's Thomas J. Watson Research Center, the Australian Computing and Communications Institute, RoSecure Software Pty Ltd, Sterling Software, and currently with Qualcomm Australia. Interested in the issues of distributed system administration, including network management, and development of tools for more effective and cost-effective use of network environments. Also security and privacy considerations, including use of cryptography.

References

- [1] *Protect Your Privacy*. Stallings, W. Prentice Hall PTR, 1995.
- [2] Kantor, B., and Lapsley, P. *Network News Transfer Protocol*. RFC 997, February 1986.

The Brave Little Toaster Meets Usenet[†]

Karl L. Swartz – Network Appliance

ABSTRACT

Usenet volume has been growing exponentially for many years; this growth places ever-increasing demands on the resources of a netnews server, particularly disk space – and file system performance – for the article spool area. Keeping up with this demand became substantially more difficult when it could no longer be satisfied by a single disk, and since netnews is incidental to SLAC's research mission, we wanted to find a solution that could easily scale to meet future growth while requiring minimal system administration effort. In the process of evaluating the various solutions that were proposed, we developed benchmarks to measure performance for this specialized application, and were surprised to find that some of our beliefs and intuition were not supported by the facts.

The alternatives considered by SLAC are described, as are the benchmarks developed to evaluate the alternatives, and the results of those benchmarks. While netnews is the application we examined, our experience will hopefully provide inspiration for others to more carefully evaluate their applications instead of using stock benchmarks that may not correlate well with the intended use. Our results may also break down some biases and encourage the reader to consider alternatives which might otherwise have been ignored.

Introduction

This paper discusses work begun while the author was employed by the Stanford Linear Accelerator Center (SLAC).

In 1992, Usenet had outgrown the then-current netnews server at SLAC. A study of growth trends, based on data gathered from our server and other sources, provided forecasting tools to guide the configuration of a server that would be adequate until early 1995 [1]. Despite an unprecedented surge in growth between mid-1993 and early 1995 [2], that server managed to outlive its original design life with minimal tinkering, lasting until mid-1995 before suffering a major collapse due to netnews volume.

Even before that collapse, SLAC had been studying what to do for a next generation server. The laboratory's primary mission is research in high-energy physics and related fields, so the ongoing drain of resources for the care and feeding of an incidental service like Usenet was becoming an irritant, especially in a time of shrinking budgets and layoffs. Four goals were established to guide specification of the new server:

- Capacity and performance to accommodate projected growth in traffic and readership thru 1997, without reducing newsgroups or expiration times.
- Simplicity of future growth.
- Greater reliability.
- Reduction in administrative labor costs.

The capital equipment budget for this project was hardly munificent, imposing yet another constraint. (This was eased somewhat after it was shown that the initial budget would not even pay for the necessary disks.)

The key to this problem was the second point, simplicity of future growth. Most of the reliability shortcomings of SLAC's earlier netnews servers had come as they neared their design limits and became increasingly susceptible to collapse when a minor surge in traffic overwhelmed their strained resources. By mid-1995, shutting down building power or networking for maintenance over a weekend could result in the netnews server spending several *weeks* to work through the resulting backlog. Nursing and tuning such a sickly server just so it could do the job was a major consumer of labor at times. For the holiday shutdown, netnews – along with mail and payroll – was deemed a critical system which would be fixed immediately, instead of waiting until after New Year's Day, because of the cost of recovering from a protracted outage.

Critical Server Resources

The core of a netnews server consists of two large databases located on disk: the articles themselves; and the history file, which tracks which articles have been received and when they should be removed. For a full news feed and a given set of expiration times, the size of each is roughly proportional to the number of articles accepted per week. From 1984 until mid-1993 this value grew at a remarkably consistent rate of approximately 67% per year, or doubling every 15 to 16 months [1, 3]. For nearly two years starting mid-1993, growth surged to a 100% annual rate before

[†]This work supported by the United States Department of Energy under contract number DE-AC03-76SF00515, and simultaneously published as SLAC PUB-7254.

dropping back to the historic curve and perhaps even lower [2].

This history file is still relatively manageable despite exponential growth – with the generous expiration times used by SLAC, the history file and associated indexes will only require about 763MB at the end of 1997¹. The main problem is that in older versions of the dbz library used by C News and INN, the `dbzagain()` routine did not automatically reduce the size of the tag stored in the `history.pag` file as the history file grew beyond the 2ⁿ bytes initially planned for. SLAC discovered this when investigating why `expire` was taking over two days to run – it seemed to be trying to keep the entire history file in memory and was thrashing badly. Rebuilding the history dbz index provided a quick fix until updated software could be installed to keep the problem from recurring as the the relentless growth continued.

Unfortunately, the article spool area is a far more difficult beast to tame. Using the SLAC server as an example again, 8.9GB will be needed at the start of 1997, growing to 14.9GB by the end of the year. Even the later, larger size wouldn't be too bad if not for the fact that it will consist of nearly 4.7 million files, with over 1.8 million new files being created (and nearly as many deleted) each week. The *average* rate is three file creates and deletes per second – greater capacity is needed to handle short-term surges. If that isn't bad enough yet, the problem gets worse if the articles are not all stored in a single file system.

The reason for this is that the structure of news is mapped directly onto the file system. The hierarchical nature of newsgroup names becomes a directory hierarchy, and each article is stored in its own file in the directory corresponding to its newsgroup. Cross-posted articles are implemented with links – preferably hard links, though symbolic links are used if necessary. This is the reason for wanting to have the entire article spool in a single file system, since a hard link requires just another directory entry, while a symbolic link imposes yet another file creation (and eventually deletion) along with hits on two file systems when referenced.

Using multiple file systems also forces the news administrator to invest effort in guessing how to allocate newsgroups to the available file systems in a manner which balances both space and load. It may be difficult to change this allocation later, and a good balance now may not be good in the future if one set of groups grows faster than another. Managing such a setup is an intractable problem.

¹SLAC uses 17 days as the expiration period for essentially all newsgroups and for history data. History data had been kept for 30 days, but when disk and memory constraints became severe it was decided that this no longer added much value given the fast propagation of netnews in the net today.

Large File System Alternatives

SLAC's netnews server was using two file systems for the article spool so we were all too familiar with the problems with that solution. It was fairly clear that a single, large disk would probably be a problem for performance, even if we could get one that was large enough (9GB, the largest readily available disk when the system was being acquired, would work, but not even into 1997) and access it as a single file system (with SunOS, we were limited to a 2GB file system). That would still leave the requirement for simplicity of future growth unaddressed.

To get a single, large file system, Sun's OnLine: DiskSuite [4] appeared to be the answer. Our netnews server was a Sun (running SunOS 4.1.3) and other sites seemed to be successfully using it for news. This product increases the maximum size of a file system on SunOS from 2 gigabytes to 1 terabyte. It allows the creation of large volumes via striping ("RAID 0") or non-interleaved concatenation of multiple disks. It also offers the option of higher availability and reliability through mirroring (RAID 1) [5] and hot spares.

The choice between organizing the disks as a concatenation or a stripe set was difficult. Striping would seem to be better for performance since it spreads data over all disks. A superficial analysis of concatenation suggests it would fill most of one disk before moving on to the next one. However, the BSD Fast File System creates directories "in a cylinder group that has a greater than average number of free nodes, and the smallest number of directories in it," then tries to place files close to where their directory is located [6]. With the large number of directories in the article spool, one would expect data to be spread amongst disks fairly quickly.

The weakness of RAID 0 is that the size of the stripe set is fixed when the RAID virtual device is created, whereas a concatenation can be expanded as needed. A file system on a stripe set can be expanded by concatenating another stripe set, but that may mean buying more disks than are required for the desired capacity. With the price of disks always dropping (while capacity and performance increase), buying disks well ahead of need is not appealing. An alternative is to concatenate just one or two disks, but that raises the same performance concerns that motivated the choice of striping in the first place. Why not just start with a concatenation?

Many system managers claim that holes in an NNTP stream are more valuable than the data. [7]

While many might debate the value of most netnews content, there's no doubt that a file system composed of multiple disks in which there is no redundancy – wherein a single drive failure can cause the loss of all file data – is not a step towards the goal of greater reliability. It also adds to administrative costs because a failure becomes a crisis instead of a

nuisance which can be resolved at relative leisure. With OnLine: DiskSuite, mirroring (RAID 1) is the only available solution, possibly with one or more hot spares to even further reduce the urgency of a failure.

Mirroring unfortunately requires twice as many disks, preferably spread over twice as many controllers. With a requirement of 14.9GB, and using fast 4GB disk drives, eight drives and two controllers would be needed. Another year's growth would require two more controllers, forcing us to consider a more expensive SPARCserver 20 instead of the SPARCserver 5 we were contemplating, just to get enough SBus slots. The hardware costs were escalating at an alarming rate! The only bright spot was that the ability to choose from amongst several disks when reading articles might mean a mirrored file system would perform better, assuming reads account for a significant percentage of the requests and the overhead of mirroring doesn't overwhelm this advantage.

RAID 5 would certainly have reduced the hardware investment, but DiskSuite didn't have it until the Solstice DiskSuite 4.0 release [8]. This would have required at least Solaris 2.3, and with limited resources, SLAC had not yet taken on the challenge of supporting Solaris 2. Other vendors were also undesirable because we hoped not to invest the effort in migrating our netnews service to a whole new operating system.

Around the time of this design effort, SLAC's High-Performance Computing Team (known informally as the "Farm Team") was looking at various large, high-performance file systems for use in a prototype data analysis effort. One of the alternatives being explored was a Network Appliance filer (commonly known as a *toaster* because of its appliance-like simplicity), and it was suggested as an appealing solution to the netnews problem. This was not the first time this product had been considered for netnews at SLAC – the large file system was very appealing – but the prospect of NFS achieving adequate performance relative to local disk for the many small files involved in processing netnews seemed far-fetched.

There weren't any other appealing ideas on the horizon, so with some trepidation, we looked at the Network Appliance product further. In addition to the large file system, it had several other appealing features. More disk drives could be added as needed, even a single disk of different geometry from the others, without impacting performance.

Even more interesting was the addition of support for very large directories [9] since very high volume newsgroups produce directories with many thousands of files in them. Processing all files in a directory of n files requires $Order(n^2)$ search time in a traditional UNIX file system. BSD 4.3 reduces this to $Order(n)$ for programs that process files in sequential, directory order [10], but netnews often accesses files in a manner which reduces the effectiveness of this

optimization. The BSD solution also does not help file creation, which is also $Order(n)$ (i.e., $Order(n^2)$ to populate a directory of n files). The Network Appliance design only needs to examine $n/256$ directory entries in the file creation case, and by using hash signatures even fewer string comparisons are required. This is still technically $Order(n)$, but with a much smaller constant multiplier it should be considerably faster. (Processing all files in a directory in non-sequential order is similarly still $Order(n^2)$ but with a smaller multiplier.)

We agreed to at least give an NFS solution from Network Appliance a chance, and they supplied a NetApp 1400 for evaluation.

Benchmark Specification

Standard benchmarks can be useful tools for comparing the performance of products from different vendors for common uses. Their usefulness is diminished when one is confronted with a very specialized application. A netnews server places unique and demanding load patterns on a file system, so standard benchmarks were not considered for more than a moment in evaluating NetApp's filer. Besides, the obvious benchmark would have been SPEC SFS (LADDIS) [11] which only tests NFS. Thus it could not have been run against a local file system, one of the two alternatives we were considering. We therefore set about devising a suitable application benchmark which could be run against both alternatives and which would provide data which could be clearly correlated to an actual netnews server.

The first step was to identify the key activities of a netnews server. Four such activities were identified.

- Receiving and storing new articles.
- Sending articles to other sites.
- Expiring old articles.
- Serving articles to readers.

Receiving and storing new articles consumes the majority of most netnews servers' time. There is no opportunity for parallelism, even in the presence of multiple news feeds, because the incoming article streams are effectively serialized before checking the history file and possibly updating it and storing the article. The goal of a server able to support the load expected at the end of 1997 means the server must be able to process each article in less than a third of a second.

One of the challenges is that adding a file to a directory is an $Order(n)$ problem in a normal UNIX file system, as discussed above. Each new article requires a file creation, and possibly the addition of links to other directories if the article was cross-posted. Typical user directories rarely have more than a few hundred entries, and adding files to them does not significantly impede other activity on the system. Unfortunately, neither property is true for netnews.

Sending articles to other sites was not initially perceived as being an important activity with regard to the file system, since nearly all of SLAC's outgoing news feeds use NNTP, and INN² [12] tries to send articles as soon as they are accepted, which means they come from buffer cache and not from disk. David Lawrence noted that UUNET had encountered problems catching up when downstream NNTP sites went down. Reading the articles back from disk turned out to be a significant bottleneck [13].

Much of the process of expiring old articles happens in parallel with other netnews processing, but if `expire` doesn't get rid of old articles fast enough, incoming news may be stalled until sufficient space is available. Traditionally, `expire`'s file deletion pattern exhibits the $O(n^2)$ behavior of pathological cases.³ While a command like `rm` will process files in the order they appear in a directory, taking advantage of BSD's optimizations, `expire` generates delete requests within a given directory in the order the files were created. With previous expirations plus article cancellations and other activities creating many holes in a directory, creation order may end up being fairly random. `Expire` also tends to jump from one directory to another, rather than focusing its efforts on one directory before moving on, which not only may cause disk seeks but also causes BSD to flush the cache which it uses to improve sequential directory accesses.

Fortunately, INN includes the `fastrm` program which avoids these and other shortcomings of older `expire` implementations. INN's `expire` merely generates a list of files which it wants to delete, and feeds this list to `fastrm` to do the deletion.

Finally, serving articles to readers would seem to be a very important part of what a netnews server does. In terms of performance, given a modest reader population, this task is in fact of little consequence. Multiple readers can be served in parallel, and any given reader most likely won't mind if fetching an article takes half a second instead of a quarter of a second. Such delays won't cause a backlog of work to pile up, at least for the server.

The odds are that most articles won't ever be read anyway. Consider that at the time of LISA X, SLAC's netnews server is expected to be accepting about 900,000 articles per week. SLAC has roughly 1,200 employees and if half of them read netnews, each will have to read an average of 1,500 articles per week – 1,500 *different* articles from those read by anyone else at SLAC – for all of the incoming articles to be read. With a much larger user community, such as

at a large Internet Service Provider, all of the articles might be read, but not at a place such as SLAC.

With these guidelines in mind, a benchmark was constructed which would measure the performance of receiving articles and of expiring them. Batching articles was added to the test set later on, after the need became more apparent.

Benchmark Construction

The simplest way to construct a meaningful and repeatable benchmark seemed to be to capture a snapshot of a news feed, then feed it into actual netnews software, timing key parts of the process. While the new server was expected to run INN, C News⁴ was chosen for the benchmarks as its many pieces seemed better suited to isolation and individual examination than the monolithic structure of INN. The data structures on disk are identical and are manipulated in similar ways, so for a study of file system performance, results from one should apply to the other.

The benchmark is a simple shell script performing some setup work, then multiple passes consisting of three phases: `unbatch`, `batch`, and `expire`. The critical piece of each phase is invoked with `time` to collect elapsed time and other statistics.⁵ Only the core program in each phase is actually used, since the locking and other overhead of the higher-level scripts would serve no purpose and would obscure the results that are of interest.

The `unbatch` phase of the benchmark, which measures the receiving and storing of new articles, is fairly straightforward. First, a large number of batches was copied to the incoming spool area. Enough data was used to ensure that all caches were flushed so as not to mask the performance of the underlying file systems. Then, the `relaynews` program was invoked directly on the batches. The same arguments were used as the `newsrun` script would use in a live system, except that no `stale` value was specified since the batches would likely contain very old news by the time the final benchmarks were run.

The `batch` phase measures sending backlogged articles to other sites. To generate the batches, C News was configured to feed a single downstream site with the following `sys` file:

```
# what we'll accept
ME:all

# downstream - everything (almost)
downstream:all,!junk:f:
```

²SLAC was still running C News at the time, even though it was fairly clear that INN was better suited to SLAC's needs. With resources scarce, installing the new software had been deferred until the new server was acquired.

³Netnews is exceedingly good at finding and exploiting pathological cases to greatest disadvantage.

⁴The "Cleanup Release of C News, with patch C.R.E.," from January 1995.

⁵`Nfsstat -c` is invoked before and after the timed piece to also capture NFS statistics. This data has not yet been analyzed as it doesn't directly impact the results of this project. It may provide some interesting tuning information, though.

In each pass, the batch generated by the unbatch phase is moved aside. Actual batching is only done every third pass in an attempt to reduce the already lengthy test time without losing too many data points, and the batch generated in the *previous* phase is used in order to simulate a seriously backlogged feed. (Since datasets were chosen to exceed memory size and thus eliminate cache effects, this probably has no real effect.) Again, only the component of the batching process which is directly affected by file system performance was used, in this case the `batcher` utility with input from the batch file and output to `/dev/null`.

The `expire` phase is a little trickier since file deletion is rolled into one program along with scanning and rebuilding the history file, at least in the C News version. Moreover, `expire` works by looking at article timestamps, which for this benchmark are totally meaningless.

Instead of using `expire`, its actions were synthesized in simplified form. First, the number of history entries was recorded after each unbatch phase. A simple awk script then uses this information to scan the history file, expiring articles from more than *retention* previous passes. The list of articles to be deleted is written to one file while the modified history file is written to another. (History entries are kept for the duration of the benchmark.) The new history file is then run thru `dbz` and moved into place.

To measure the file system component of the expiration process, the file containing the list of article files to be deleted is then fed into either `dumbrm` or `sort|fastrm`. `Dumbrm` is a simple C program which reads pathnames from `stdin` and deletes them, generating the same `unlink()` calls as `expire` itself would. `Fastrm` is the utility from INN⁶, invoked with the same options used by INN's `expire`. The `sort` is included in the timing to produce a more fair comparison to `dumbrm`; it's debatable whether or not this should have been included.

All the News That's Fit to Test

Collecting data to fuel the benchmark was accomplished by creating a fake uucp feed of all articles on SLAC's netnews server and capturing approximately half a week's batches from August, 1995. A total of 2759 batches were collected, containing 300,150 articles in 911 megabytes.

While this was good enough to generate interesting results, a set large enough to represent at least a week was desired. A script was written which reads an existing batch and modifies message ids in `Message-ID` and `Supersedes` headers and the target message id of `cancel` control messages. To do this, a delimiter and a clone number are appended to the

host portion of each message id, a combination unlikely to conflict any real message id. Byte counts were fixed and the clone batch written.

With the original data and two clones, over 900,000 articles were available, approximating the expected feed for the first week of October, 1996. The benchmark was configured to run 14 passes with 592 batches per pass (shortchanging pass 14 by 11 batches), with an expiration step of 6. This simulates a system running `expire` twice per day, with a three day retention period for all newsgroups. Most sites probably only run `expire` once per day, but the more frequent expirations encourages rapid fragmentation of the file system and directories, thus simulating a more mature system.

Testbed Configuration

The initial testbed used at SLAC consisted of a SPARCserver 2 and a NetApp 1400. While this setup produced sufficient results to make an unequivocal choice between the alternatives, the author's move to a job at Network Appliance offered the opportunity to run a more thorough set of tests on a wider variety of configurations. These results are the ones presented in this paper.

The primary test host used was an Axil 311, a SPARCserver 20 clone but with a CPU module that appeared to be equivalent to that in a SPARCserver 10/41. This system was equipped with 128MB of memory, a Sun Quad Ethernet interface card, and a Cisco CDDI interface card. Five 4GB Seagate ST15230N (Hawk) Fast SCSI disks were attached to the on board SCSI bus, with the first used as the system disk (the internal disk was disconnected) and the other four used for `/var/spool/news` (the article spool file system(s)) or unused when testing against a filer. SunOS 4.1.4 was installed with a large (nearly 2GB) partition left on the system disk for `/usr/lib/news` (where the history file and news configuration files are stored). C News binaries and the benchmark itself were also stored on this disk.

Two filers were used, both running the NetApp 3.1.4c Data ONTAP software release. The first was a NetApp 1400 configured identically to the one purchased by SLAC - 128MB, 2MB NVRAM, a single 10 megabit/second Ethernet interface, and seven 4GB Seagate ST15230N Fast SCSI disks. (This model is no longer offered; the current entry-level NetApp F220 is about twice as fast as the NetApp 1400 based on LAD-DIS results.) To explore the performance of a high-end filer which a large Internet Service Provider might prefer for greater performance and/or disk capacity, a NetApp F540 was also tested. This filer had 256MB, 8MB NVRAM, both 10/100 megabit/second Ethernet and CDDI interfaces, and seven 4GB Seagate ST15150W (Barracuda) Fast/Wide SCSI disks.

There was some debate about the effect of several filer options recommended by Network Appliance

⁶INN1.4-sec from December 22, 1993.

for netnews applications, as well as the value of FDDI versus a dedicated Ethernet, so another Axil was borrowed to run some abridged test runs against the filers while the first Axil was running various test runs involving only local disks. This second unit was an Axil 235, apparently a SPARCserver 10 clone with a SPARC 20 CPU module (!), with 64MB, a Cisco CDDI interface card, and a single 4GB Seagate ST15230N Fast SCSI disk that was cloned using `dd` from the first system after it had been configured but before OnLine: DiskSuite was installed.

Other than one of the filer software configuration options (`no_atime_update`), default values were used on the filers and on SunOS. In particular, extra inodes were not allocated on either the filers or on SunOS file systems during initialization, nor was the Snapshot feature of the filers [14] disabled. The default inodes value for `newfs` seemed to be adequate. The filers would need more inodes in practice, but since the number of inodes can be increased on-the-fly there's no need to add more until one has a better idea of how many will be needed. The first impulse would be to disable Snapshots entirely, but having at least one hourly Snapshot might be handy for recovering from the occasional slip of the fingers as root. Since the cost of creating a Snapshot is inconsequential, there's no performance reason to disable them, only the need to recover disk space fairly quickly after `expire` deletes articles.

The Ethernet consisted of a crossover (hub-to-hub) cable for the Ethernet tests, and an isolated DEC CDDI concentrator for the FDDI tests. Another machine was also attached to the FDDI network to provide a repository for the test batches and for test results. (No access to this machine took place during instrumented portions of the benchmark.)

disk	type	hierarchy
1	mnt	alt (alt.binaries linked to disk 4)
2	link	rec,soc,talk,de
3	link	comp,misc,sci,news,bit,gnu,vmsnet
4	dir	everything else (incl. alt.binaries)

Figure 1: Newsgroup allocation for symlinked local disks and method of attaching hierarchy to main spool area.

The benchmark runs using the four local disks for `/var/spool/news` were done with three different configurations. The first was run without OnLine: DiskSuite and used a 2GB partition on each disk⁷, with space parceled out by hand via symlinks (and a mount for `alt`) as detailed in Figure 1. For the next test, OnLine: DiskSuite was used to create a 16GB striped partition using an interlace factor of

16KB.⁸ The third configuration used two concatenated disks, with the second pair of disks mirroring the first pair, providing an 8GB file system.

Toaster or Local Disk?

The results of the benchmark runs were dramatic. Because of the added operational flexibility of the toaster, we would have been happy if its performance was comparable to local disks. In fact, as Figure 2 shows, the 1400 was nearly four times as fast, on average, as the best configuration of local disks on the unbatch tests. The F540 was over five times as fast! Equally surprising was the poor performance of the OnLine: DiskSuite configurations. The mirrored concatenation arrangement was barely 50% faster than the absolute minimum of three articles per second required by the end of 1997, precious little headroom for catching up much less capacity for future growth.

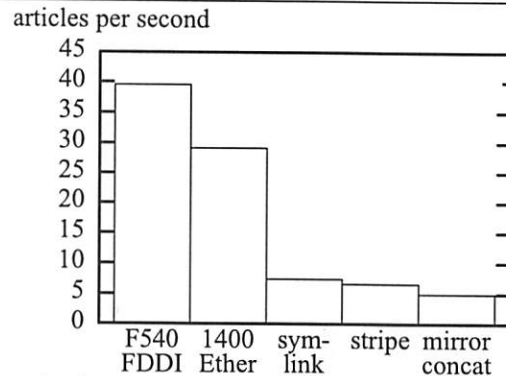


Figure 2: Average number of articles unbatched per second over full benchmark run for various configurations.

The only result that didn't come as a big surprise was that striping was faster than concatenation (with mirroring), by about 33%. The conjecture that concatenation might cause one disk (or mirrored pair) to be filled before moving on to the next was not borne out, however. During the runs, visual observation of the disk activity lights and monitoring via the `iostat` utility both indicated that the data was being spread amongst the two disks even though neither was near capacity.

Looking at the data in more detail, some performance degradation can be seen in Figure 3 as the file system fills and ages. In pass 14, the NetApp F540 and the symlinked local disks both retain 86.5% of their performance from pass 1; the NetApp 1400 lost a bit more (down to 85%) but the difference is probably not significant. Pass 8 is the first pass after expiring articles (only articles more than six passes old are

⁷2GB is the largest partition supported by SunOS 4.1.4 without using OnLine: DiskSuite.

⁸The default for OnLine: DiskSuite is the size of a cylinder on the first disk, which seemed inappropriate for a modern SCSI disk for which all cylinders might not be the same size. The 16KB value was borrowed from the default in Solstice DiskSuite 4.0.

expired, so the expire phase at the end of pass 7 is the first to actually do anything), which presumably has something to do with the surge in filer performance. The local disk test shows a similar though less pronounced effect, with 7.1% better performance in pass 8 than in pass 7, compared to 12.6% for the NetApp 1400 and 14.8% for the NetApp F540. This may be an artifact of the test data, but no examination has been done to determine if there is anything unusual about the articles processed in pass 8.

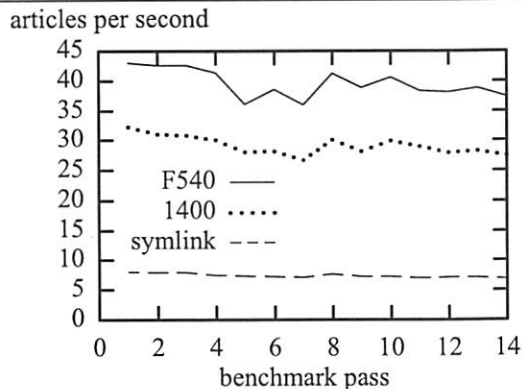


Figure 3: Articles unbatched per second

The performance advantage of the filers over local disks is even more dramatic for expiration. As seen in Figure 4, the F540 is over an order of magnitude faster than local disk and the 1400 is a respectable 8.5 times as fast. Data for the OnLine: DiskSuite configurations is not shown because each pass was painfully slow and getting slower. With limited equipment time available, the mirrored concatenation test was stopped after six passes, which were sufficient to show that it was significantly slower than the symlinked arrangement, while the stripe run was interrupted after five complete passes by a power failure.

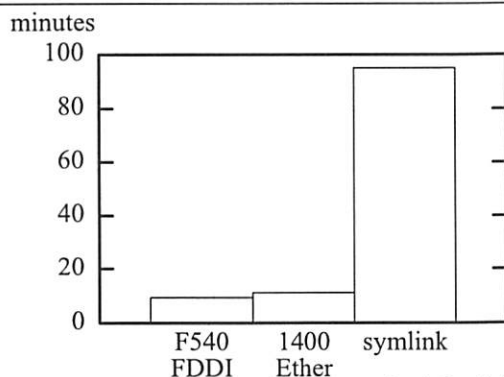


Figure 4: Average time in minutes required for delete portion of expire.

All of the tests in this set used *dumbrm* rather than INN's *fastrm*. This made little difference for the filers, as detailed later in this paper. Presumably expiration on local disks would have benefited greatly from *fastrm*, but the unbatch numbers had already convincingly shown that local disks were marginal at

best for the job, so it was felt that the several days of test time needed to complete a *fastrm* run would not be productive.

The final set of comparisons are those for batching outgoing news feeds, shown in Figure 5. While the difference is not as dramatic as it is for the other parts of the benchmark, the filers are still faster than local disks. In addition, the local disk results appear to be getting slower as the file system ages, whereas the filers suffer much less performance degradation.

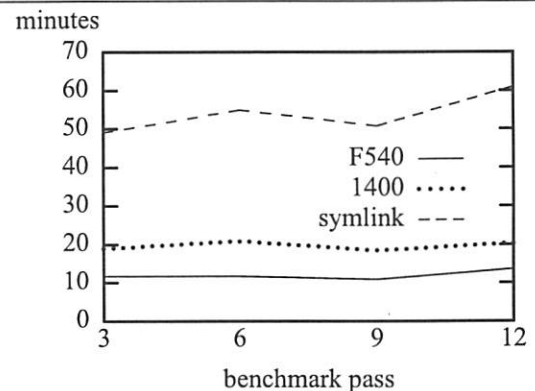


Figure 5: Average batch time in minutes.

Light or Dark Toast?

Prior to the final set of filer runs described above, another set of tests was run to evaluate the effects of several alternative filer and networking options. Network Appliance recommended that the following filer options be changed from the defaults for netnews applications:

```
options no_atime_update on
options minra on
```

The first option causes the filer to not update access times on files. For netnews, there's no apparent value in maintaining file access times, and not updating them saves the expense of writing the updates to disk.

The case for the second option, which causes the filer to refrain from aggressive read ahead (in anticipation of sequential access to an entire file), is less clear. Assuming all user access is via NNTP [15], the only apparent cases in which one would start reading part of an article and not read to the end-of-file would be using NNTP's HEAD command or if the connection is broken during an ARTICLE or BODY command. With contemporary newsreaders using the XOVER command and the NOV database to access most data formerly obtained using HEAD, it's not clear what value turning off read ahead provides. The only plausible justification is that Internet Service Providers who might have large numbers of customers accessing large binary postings, via comparatively slow modem links, might end up wasting filer memory, and perhaps causing thrashing, because too much data is being cached too far in advance of its being needed.

Besides studying these options, we wondered how much benefit would be derived from the lower latency of FDDI, the benefit of *fastrm* on the filer, and whether or not it would be advantageous to also place the history file on the filer.

A series of benchmark runs were done using the NetApp F540 and the second Axil, described above, to compare these alternatives. Since there were a number of different runs to perform, they were abbreviated to only ten passes. This allowed three batch samples and three post-expire unbatch samples, which was felt would provide enough data to draw reasonable conclusions without taking an inordinate amount of time.

The first pairing was FDDI versus Ethernet. Using FDDI, the unbatch tests ran in an average of 90.8% of the time needed with Ethernet. Batching was even faster, taking only 88% of the Ethernet time. Expire was marginally slower over FDDI, but the difference is probably statistically insignificant. The remainder of the tests with the NetApp F540 were done using FDDI. (Tests of the NetApp 1400 were done using Ethernet because it did not have an FDDI interface.)

The next set of tests individually compared the two recommended options against the baseline FDDI test. Neither had any significant effect on unbatching, which was unsurprising since both options influence reads, and unbatching does little reading from the article spool. For batching, *minra* had little effect (it was expected that it would hurt) but not updating access times saved about 5% of the baseline FDDI time. As expected, expire saw no benefit from not updating access times, but seemed to be slightly faster with *minra*. It's not clear why this option would have any effect on file deletion. Since only *no_atime_update* was clearly beneficial it was the only option used in the remainder of the tests.

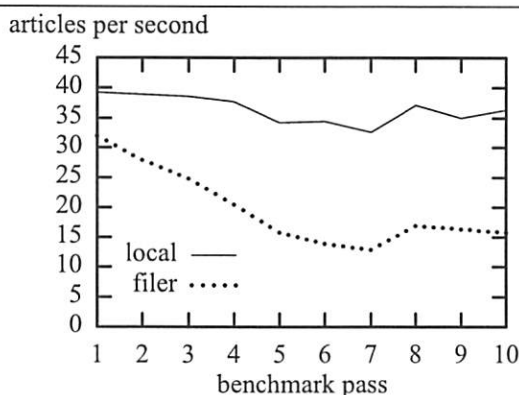


Figure 6: Average number of articles unbatched per second with the history file on the filer instead of local disk, with articles stored on the filer in both cases.

The next test showed that expire ran several percent faster using *fastrm*. However, since we did not

have *fastrm* data for the local disk cases in the primary testbed, the full benchmarks run against the filers were done using the marginally suboptimal (for filers) *dumbrm*.

The last of these tests produced the most interesting results. With the history file and other contents of `/usr/lib/news` on the filer along with the article spool, batching was about 2% slower and expire about 3% slower than only having the article spool on the filer. No difference would have been expected for expire since the timed portion of expire does not access anything in `/usr/lib/news`. Presumably the extra filer activity before this step pushed some data out of cache, slowing expire. The big difference came during the unbatch tests. As illustrated in Figure 6, putting the history file on the filer produced unbatch times which started off taking 22% longer, progressing to as much as 150% more time as the system aged.

Future Research

One of the implementors of the log-structured file system (LFS) in BSD 4.4 [16, 17] suggested that netnews would be a good application for LFS, since a log-structured file system is designed to optimize writing to disk. This is how the critical unbatch part of netnews processing spends much of its time. The success of the Network Appliance filers is consistent with this conjecture since their Write Anywhere File Layout (WAFL) [14], while not a log-structured design, similarly optimizes writes to minimize head seeks. Using a BSD 4.4 system to compare an LFS-based article spool file system to an FFS-based equivalent would be interesting, though a filer would still be expected to offer better performance due to the large directory support, if nothing else.

The b+ tree data structures used for directories in the Windows NT file system [18] to allow it to perform quick file lookups in large directories make NTFS seem appealing for netnews. Other aspects of NTFS appear to incur more overhead than would be desired for an application as demanding as netnews. Since Network Appliance's new Multiprotocol Filer software features native support for CIFS, the equivalent of NFS in the Windows networking world, and several netnews implementations are available for Windows NT, another netnews comparison project is likely in the author's future.

Conclusions

*Then let us praise the brave appliance
In which we place this just reliance* [19]

The surprisingly fast performance of the Network Appliance filer in the netnews benchmarks made the decision regarding SLAC's netnews server obvious. More important, though, the results served as a strong reminder to avoid preconceptions. Benchmarks can produce surprising results, which presumably is why many people run them in the first place. Finally,

NFS, despite its age and weaknesses, can still do a remarkably good job.

Availability

The benchmark tools may be made available if there is interest. Please contact the author via e-mail at kls@netapp.com for more information.

Acknowledgments

Special thanks to Mom and to my wife, Krissie. Mom was always encouraging even when she had no idea what I was really working on. I'll miss her. Krissie has been very understanding and supportive of my long work hours during what is supposed to be our honeymoon year. Others who helped in various ways include Mark Barnett, George Berg, Chuck Boenheim, Bob Cook, Renata Dart, Rosemary Dinelli, Walt Disney Co., Guy Harris, Dave Hitz, Moana Kutsche, Randy Melen, Lincoln Myers, Rob Salmon, Arnie Thompson, Andy Watson, Bebo White, and others whose contribution is not diminished by my failure to remember them here. My gratitude goes to all of them. Thanks, too, to Alexander for his patience. Still no more skunks, but he's on Bath Row anyway.

Author Information

Karl Swartz was Team Leader of the System Administration Team in SLAC Computing Services at the Stanford Linear Accelerator Center when this work was started. He was so impressed by the toaster's performance that he joined Network Appliance as a Technical Marketing Engineer. Prior to SLAC, he worked at the Los Alamos National Laboratory on computer security and nuclear materials accounting, and in Pittsburgh at Formtek, a start-up now owned by Lockheed-Martin, on vector and raster CAD systems. He attended the University of Oregon where he studied computer science and economics. Between work and a new wife, Karl hasn't been on the racetrack in far too long, but he does find time to moderate a newsgroup (sci.aeronautics.airliners) and to enjoy good food and good beer and trips to the beach with his wife, Krissie, and their Golden Retriever, Alexander. Krissie would be very upset if either of them castrated or slaughtered cattle. E-mail Karl at kls@chicago.com or kls@netapp.com.

References

1. Karl L. Swartz, "Forecasting Disk Resource Requirements for a Usenet Server," *Proceedings of the 7th USENIX Large Installation System Administration Conference (LISA VII)*, pp. 101-108, Monterey, California, November 1993. Also published as SLAC-PUB-6353.
2. Karl L. Swartz, "Usenet Growth Graphs," <http://www.chicago.com/~kls/news-growth.html>.
3. Rick Adams, Usenet post, c. September 1993.

4. *OnLine: DiskSuite Reference Manual*, Sun Microsystems, Mountain View, California, 1991.
5. D. Patterson, G. Gibson, and R. Katz, "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *ACM SIGMOD 88*, pp. 109-116, Chicago, June 1988.
6. Marshall Kirk McKusick, William N. Joy, Samuel J. Leffler, and Robert S. Fabry, "A Fast File System for UNIX," in *4.4BSD System Manager's Manual*, O'Reilly & Associates, Sebastopol, California, April 1994.
7. V. Jacobson, "Compressing TCP/IP Headers for Low-Speed Serial Links," *RFC 1144*, February 1990. Footnote 29.
8. *Solstice DiskSuite 4.0 Administration Guide*, Sun Microsystems, Mountain View, California, March 1995.
9. Byron Rakitzis and Andy Watson, *Accelerated Performance for Large Directories*, Technical Report 3006, Network Appliance, Mountain View, California, February 1996.
10. Samuel J. Leffler, Marshall Kirk McKusick, Michael J. Karels, and John S. Quarterman, *The Design and Implementation of the 4.3BSD UNIX Operating System*, Addison-Wesley, Reading, Massachusetts, 1989.
11. Mark Wittle and Bruce E. Keith, "LADDIS: The Next Generation in NFS File Server Benchmarking," *Proceedings of the 1993 Summer USENIX Technical Conference*, pp. 111-128, Cincinnati, Ohio, June 1993.
12. Rich Salz, "InterNetNews: Usenet transport for Internet sites," *Proceedings of the 1992 Summer USENIX Technical Conference*, pp. 93-98, San Antonio, Texas, June 1992.
13. David Lawrence, private conversation, January 1996.
14. Dave Hitz, James Lau, and Michael Malcolm, "File System Design for an NFS File Server Appliance," *Proceedings of the 1994 Winter USENIX Technical Conference*, pp. 235-245, San Francisco, January 1994. Also published as Network Appliance Technical Report 3002.
15. Brian Kantor and Phil Lapsley, "Network News Transfer Protocol," *RFC 977*, February 1986.
16. Margo Seltzer, Keith Bostic, Marshall Kirk McKusick, and Carl Staelin, "An Implementation of a Log-Structured File System for UNIX," *Proceedings of the 1993 Winter USENIX Technical Conference*, pp. 307-326, San Diego, California, January 1993.
17. Margo Seltzer, Keith A. Smith, Hari Balakrishnan, Jacqueline Chang, Sara McMains, and Venkata Padmanabhan, "File System Logging versus Clustering: A Performance Comparison," *Proceedings of the 1995 Winter USENIX Technical Conference*, pp. 249-264, New Orleans, Louisiana, January 1995.

Helen Custer, *Inside the Windows NT File System*, Microsoft Press, Redmond, Washington, 1994.

19. Thomas M. Disch, *The Brave Little Toaster*, Doubleday & Company, Garden City, New York, 1986. The full-length animated movie of the same title, based on this novella, inspired the title of this paper.

A Simple Caching File System for Application Serving

John D. Bell – Ford Motor Co.

ABSTRACT

In large installations, it can become very difficult to install many hundreds of applications on every workstation. To make every package available requires that the users' command search path be very long, or that there be 'wrapper' scripts written for every application which are maintained in a single directory. This policy also requires that each workstation have enough local disk to contain all files for every application. One traditional solution is to make applications available to workstations by serving them via NFS. However, this method can cause severe network loads, and it makes every user dependent upon the uninterrupted operation of the network.

This paper describes a set of programs designed to eliminate these problems. They provide for the automatic configuration of the wrappers needed for each application (including setting environment variables, etc.), in an architecture-independent fashion. They also implement caching from NFS-mounted "lockers" to a local file system, based upon file access patterns. This gives the advantages of single point of installation and minimization of local disk requirements, while providing the user with reduced dependence upon the network and increased speed of access. These programs require no kernel modifications nor other special facilities, and have proven to be portable across many versions of UNIX.

Introduction – Rationale for our Solution

Maintaining the set of available end-user applications is one of the most important tasks of a systems administrator. Indeed, one could argue that the applications are the justification for the computers to be used at all, and hence for the systems administrators to have any work! The evolution of the methods used to maintain these applications bears examination.

Local Files

In installations which consist of just a few dozen or less workstations, with just a few applications, it is feasible to locally install everything on every machine. However, this can quickly get out of hand. Ten applications, each with an installed footprint of 100 megabytes, would require an extra gigabyte of disk on every workstation. With only 50 workstations, if each of 10 applications is updated twice a year at one hour per upgrade (locally) per system, this would consume 25 person-weeks, or almost half of one system administrator's time. Although tools like *rcp* and *rdist* can help, configuring and maintaining the system lists and command scripts for these is time-consuming and filled with opportunity for errors. Clearly, as the installation grows in size, heterogeneity of platform types, and richness of applications, this becomes unsupportable.

NFS Files

The next obvious method for distributing applications is NFS serving of the files. This has the distinct advantage of giving one point of installation and configuration. However, files delivered over a 10-Mbit

per second Ethernet network are accessed almost one order of magnitude slower than on a local disk. Also, this solution mandates that the network be fully reliable and not over-crowded. In addition, simply distributing the files via NFS does nothing to simplify the length of the command search path; in fact, if NFS-served directories are in the *\$PATH*, the user's shell will be slowed down tremendously (or possibly completely stalled) when it hashes its lookup of executable commands.

Cached Files

This led to the solution developed at Ford. Like the NFS solution, there is a single point of installation. However, (after migration), files are accessed from a local disk, which makes it both faster and more robust in the face of transient network failures or overloads. On the down side, the system is somewhat more complex than a simple NFS server scheme. Also, since there is no callback mechanism, files are not immediately consistent everywhere. (In practice, since differences usually only occur when a new package is installed or an existing one is updated, the propagation delay is acceptable.)

How It Works

The system we use can be divided into two almost disjoint components. The first is an applications dispatcher. This takes the place of individually created startup scripts, and centralizes the configuration necessary to get the various applications to run. It also provides just one directory where all these entry points can be found. The second component is the set

of programs which maintain the caching file system, and migrate file in and out based upon access times.

'runprog'

The applications dispatcher is called *runprog*. It is a relatively small, compiled (for speed) program, which takes its configuration from a table of structures, one for each entry point supported. *runprog*, when invoked as "runprog update", automatically creates symbolic links to itself named according to the various programs configured into it. Invoking *runprog* by one of these other names causes it to dispatch to that program.

Along with the name mapping, *runprog* supports configuration options for the various programs. It can invoke the target program by the absolute (mapped) pathname, for applications that derive auxiliary files relative to the directory where they are installed. It can restrict running any configured program to just certain users or hostnames, or any except certain users or hostnames. It can set any environment variables before the target program; this can be used to set items like the PATH, the installation base directory for a package, or the shared libraries' search path. *runprog* also runs set-UID root to permit it to do usage accounting, and to permit a target program to be invoked as a particular special UID (not the user's). For example, we configure "kermit" into *runprog* as set-UID to "nuucp", so that it may manipulate the lock files for the serial communication device files it shares with UUCP. Please refer to Appendix A for a configuration file example.

The Cache

The actual file caching system consists of four components: one or more read-only NFS-served file systems (which we called "lockers"), which contain the source files; the local hierarchy (the "cache"), which contains either copies of the files in the lockers, or symbolic links to them; a program which updates

the hierarchy in the cache to reflect changes in the various locker hierarchies; and a program which migrates files into and out of the cache, based upon access times.

'rplink' Symlinks

An example should make this clearer. Here is a typical mapped directory from our system. It would be revealed by invoking *runprog* with the arguments "listd directory_name"; see Figure 1.

Here, the various GNU tools, served from the locker "red.pto", are mapped into the cache. Note that the final component of the locker pathname is equal to the architecture of the client machine. Listing the locker directory gives:

```
% ls -bCF /ford/red.pto/loc/GNU
aix/      irix6_r8k/  solaris/
irix5/    osf1/       sun4/
```

On each client, the appropriate version is mapped to a uniform pathname in the cache. When the cache is empty, listing the directory gives the output shown in Figure 2. The symbolic link *.rplink* (for "runprog link") points to the corresponding directory on the locker; the file *.rpkeep* indicates that this is a mapped directory, and hence must always exist in the cache. The contents of the locker's directory are made to appear here by indirect references through *.rplink*. It is the access through these links which triggers migration.

Keeping Caches Up-To-Date

Periodically (via *cron*), a program is run on each server against each locker to generate a list of changed files (either added, deleted, or contents or *stat()* information changed). These change logs are made available to each client in a sub-directory called "/ford/lockername/changes". At some later time (again via *cron*), each client runs a program which examines these lists for all lockers, and updates the

```
% runprog listd GNU

Name      Client Directory -> Server Directory
GNU       /ford/server/loc/GNU -> /ford/red.pto/loc/GNU/solaris
```

Figure 1: Symbolic links

```
% ls -AF1 /ford/server/loc/GNU
total 14
-rw----- 1 root devel  0 May 7 11:04 .rpkeep
lrwxrwxrwx 1 root devel 29 May 7 11:04 .rplink -> /ford/red.pto/loc/GNU/solaris/
lrwxrwxrwx 1 root devel 11 May 7 11:04 bin -> .rplink/bin/
lrwxrwxrwx 1 root devel 15 May 7 11:04 include -> .rplink/include/
lrwxrwxrwx 1 root devel 12 May 7 11:04 info -> .rplink/info/
lrwxrwxrwx 1 root devel 11 May 7 11:04 lib -> .rplink/lib/
lrwxrwxrwx 1 root devel 11 May 7 11:04 man -> .rplink/man/
```

Figure 2: Listing with empty cache

mapped directories in /ford/server appropriately. This program can merge an arbitrary number of days of change logs, so that a client that has been down or off the network will bring itself back into synchronization as soon as it updates the first time.

Migration of Files

In the second phase of the update process, each client, after reconciling its cache hierarchy with the union of the locker hierarchies, examines the access times of the various symbolic links in the cache. Since the *readlink()* call updates the access time of the symbolic link, any file or directory linked back through *.rplink* which has been accessed more recently than the link's creation date indicates that the corresponding entry should be moved into the cache. That is, the update software destroys the symbolic link and copies the directory or file that the link pointed to. Note that this is the same mechanism used by other systems which run without kernel modifications (*lfu* and *nightly* – see below). As an example, after the listing of “/ford/server/loc/GNU” above, running the *update-*

client script produces the output shown in Figure 3. The *.rplink* has moved one level deeper in the directory hierarchy, as can be seen by listing the contents of the “bin” sub-directory; see Figure 4. And now, having accessed each of these symbolic links which point to *their* corresponding *.rplink*, another cache update will bring each of these files local; see Figure 5.

The cache-updating software examines how much space is available in the destination file system, sorts the files destined for local migration by access time, and moves in the most recently used ones that will fit (allowing the file system to fill up to 95% of capacity). However, this is not a one-way process; files that are currently in the cache, but have not been accessed in a “long time” (45 days or more), have the local copy removed, and a symbolic link through *.rplink* recreated in its place.

```
# /ford/lib/runprog/bin/update-client
+[dir] /ford/server/loc/GNU/bin <- /ford/red.pto/loc/GNU/solaris/bin
+[dir] /ford/server/loc/GNU/info <- /ford/red.pto/loc/GNU/solaris/info
+[dir] /ford/server/loc/GNU/include <- /ford/red.pto/loc/GNU/solaris/include
+[dir] /ford/server/loc/GNU/lib <- /ford/red.pto/loc/GNU/solaris/lib
+[lnk] /ford/server/loc/GNU/man <- /ford/red.pto/loc/GNU/solaris/man
```

Figure 3: Listing after running *update-client* script

```
% ls -AF1 /ford/server/loc/GNU/bin
total 270
lrwxrwxrwx 1 root devel 33 May 7 16:41 .rplink -> /ford/red.pto/loc/GNU/solaris/bin/
lrwxrwxrwx 1 root devel 17 May 7 16:41 addftinfo -> .rplink/addftinfo*
lrwxrwxrwx 1 root devel 16 May 7 16:41 afmtodit -> .rplink/afmtodit*
lrwxrwxrwx 1 root devel 11 May 7 16:41 b2m -> .rplink/b2m*
lrwxrwxrwx 1 root devel 15 May 7 16:41 bdftops -> .rplink/bdftops*
lrwxrwxrwx 1 root devel 13 May 7 16:41 bison -> .rplink/bison*
    125 lines deleted
lrwxrwxrwx 1 root devel 14 May 7 16:41 zforce -> .rplink/zforce*
lrwxrwxrwx 1 root devel 13 May 7 16:41 zgrep -> .rplink/zgrep*
lrwxrwxrwx 1 root devel 13 May 7 16:41 zmore -> .rplink/zmore*
lrwxrwxrwx 1 root devel 12 May 7 16:41 znew -> .rplink/znew*
```

Figure 4: Deepening directory levels

```
# /ford/lib/runprog/bin/update-client
+[reg] /ford/server/loc/GNU/bin/addftinfo <- /ford/red.pto/loc/GNU/solaris/bin/addftinfo
+[reg] /ford/server/loc/GNU/bin/afmtodit <- /ford/red.pto/loc/GNU/solaris/bin/afmtodit
+[reg] /ford/server/loc/GNU/bin/b2m <- /ford/red.pto/loc/GNU/solaris/bin/b2m
+[lnk] /ford/server/loc/GNU/bin/bdftops <- /ford/red.pto/loc/GNU/solaris/bin/bdftops
+[reg] /ford/server/loc/GNU/bin/bison <- /ford/red.pto/loc/GNU/solaris/bin/bison
    125 lines deleted
+[reg] /ford/server/loc/GNU/bin/zforce <- /ford/red.pto/loc/GNU/solaris/bin/zforce
+[reg] /ford/server/loc/GNU/bin/zgrep <- /ford/red.pto/loc/GNU/solaris/bin/zgrep
+[reg] /ford/server/loc/GNU/bin/zmore <- /ford/red.pto/loc/GNU/solaris/bin/zmore
+[reg] /ford/server/loc/GNU/bin/znew <- /ford/red.pto/loc/GNU/solaris/bin/znew
```

Figure 5: Files local after cache update

Similar Systems

There are several other software systems which locally cache files from remote file systems. Commercially available packages includes *cacheefs* from Sun Microsystems [1], and DCE DFS from Transarc (and other vendors) [2]. Other packages developed by the Usenet community include *lfu* by Paul Anderson [3,4], and *nightly* by Hal Pomeranz [5]. Each of these will be considered in turn.

cacheefs

This is available on Sun's Solaris 2.x and Silicon Graphics' Irix operating systems, and is distributed "free" with them. It is (superficially) the simplest of the packages, and has the additional advantage of keeping files on the client continuously up-to-date with respect to the server (by a callback scheme). However, it is not available on all the architectures required at Ford, and it cannot map the contents of more than one NFS mount point into the cache. It is also not clear how many clients can be supported by one server, nor how big the cached area can be.

DFS

This product is commercially supported on a variety of architectures (however, not all of the ones found at Ford), but is an additional cost item. It can be configured to architecturally map directories (so that heterogeneous clients can refer to analogous files by the same pathname), but still cannot map multiple lockers into the cache. It is the most complex of all the schemes considered, requiring setting up a complete DCE security service (based on Kerberos) as well as multiple daemon processes on each machine. On the plus side, it was designed to support very large numbers of servers and clients, as well as a very large cached file system.

lfu

lfu has several advantages over the two packages considered above. It is available for free, and has been ported to several variations of UNIX on different hardware platforms. It appears to be able to be configured to map an architecturally-dependent locker pathname into the cache, as well as being able to serve files from more than one locker into the cache. It is considerably more complex than *cacheefs*, but quite a bit simpler than DCE. (In fact, much of the design of *runprog* was directly inspired by *lfu*, although none of the code was taken from it.)

Nightly

nightly shares many of the advantages of *lfu*; it is also available for free from "the Net", and is extremely portable (being written in Perl). It is much less complex than DFS or *lfu*. However, it does not map the client's architecture, nor can it map from multiple lockers into one cache. It is also a "smaller" solution than any of the others – in the described implementation, there are only two servers and 40 clients, the locker is only about 450 Mbytes, and the cache is only about 50 Mbytes on each client.

Description of Our Environment

The software described here has been in use at Ford Motor Co.'s Manufacturing Systems Office's Powertrain Operations for about four years. We support nearly 800 workstations in seven buildings located in three different cities in southeastern Michigan.

These workstations are running eight different versions of UNIX: Sun Microsystem's SunOS 4.1.3, Solaris 2.3, 2.4, and 2.5 on SPARC processors; Silicon Graphics' IRIX 5.3 on R4000 processors, and IRIX 6.1 on R8000 processors; and IBM's AIX 3.6 and 4.1 on RS/6000 processors. We have previously supported this system on SunOS 3, IRIX 4, and Digital Equipment Corp.'s Ultrix, and have mostly ported the system to DEC's OSF/1.

Today *runprog* supports 290 entry points, which map approximately 32 gigabytes of applications code. (Not all architectures have every application loaded.) Individual client machines have caches ranging from 400 Mbytes to 1 Gbyte.

Performance Measurements

runprog compromises the access speed of an application (from NFS disks instead of local disks) with the amount of local disk storage necessary. Although the access speed of an NFS-served disk varies tremendously with the performance of the underlying network, a few statistics may be of interest.

Four large applications were tested on a SPARCstation 10 connected over 10-Mbit Ethernet to the NFS servers. These are CAD, FEA, and other modeling applications typical of those used by design engineers in our community. In each case, the application was started from an "empty" cache (that is, one that

Application	"empty cache" time	final time	amount size	in cache
SDRC	1:05	:33.9	1588900	101670
CV	1:40	1:10	575001	134895
Mech	1:00	:43.2	127291	27367
HM	:38	:18.4	44651	15949

Table 1: Test results

contained nothing but the minimum mapped directory structure locally, with symbolic links back to the NFS locker for all sub-directories and files), and timed from the (command line) invocation until the first window was complete and not showing any "busy" indicator. This performance is typical of what would be observed when running the application from a pure NFS mount point. Then the cache was updated, and the application started again. This cycle was repeated until no further files migrated locally with subsequent cache updates. Note that these applications were not really "used", but just started – for a real user, more files would migrate locally with greater performance speedup. All sizes indicated here are in kbytes; all times are in minutes and seconds.

Summary

This paper has described a relatively simple solution to the problems associated with distributing and maintaining a large base of applications programs in a very large, heterogeneous workstation environment. The solution has many of the advantages of simply NFS-serving the files, yet optimizes its use of network resources.

Future directions for enhancement include the ability to distribute the administration further (via 'subscription list' of various packages without regard to the actual locker containing them). Also, the mechanism used to configure a particular package into *runprog* must be simplified.

Please contact the author by email at the above address for availability.

The author wishes to acknowledge the essential work of Ken Fox of Ford Motor Co. in the initial design and implementation of *runprog*, and the work of Gary Ross of Ford and Clinton Pierce of Decision Consultants Inc. in the maintenance and enhancement of the software and documentation. He thanks each of them for all their invaluable help.

Author Information

John D. Bell is a senior consulting software engineer with ASG Renaissance, where he has been on several assignments over the last nine years at a major automotive manufacturer in the Detroit, Michigan area. He studied at The American University, Case Western Reserve University, and Ohio State University, and subsequently has been programming and administrating UNIX systems for the last 13 years. He may be reached by U.S. mail at ASG Renaissance, 3000 Town Center, Suite 2237, Southfield, MI 48075, or by electronic mail at "jbell4@ford.com".

References

- [1] *cachefs* by Sun Microsystems, Inc. A white paper is available at <http://www.sun.com/sunworldonline/swol-08-1996/swol-08-sysadmin.html>.

- [2] DCE DFS by Transarc. A white paper is available at http://www.transarc.com/afs/transarc.com/public/www/Public/ProdServ/Product/Whitepapers/OSF_Whitepapers/dfs.ps.
- [3] Anderson, Paul. "Effective Use of Local Workstation Disks in an NFS Network", *USENIX LISA VI Conference Proceedings*, 1992.
- [4] Anderson, Paul. "Managing Program Binaries in a Heterogenous UNIX Network", *USENIX LISA V Conference Proceedings*, 1992.
- [5] Pomeranz, Hal. "A New Network for the Cost of One SCSI Cable: A Simple Caching Strategy for Third-Party Applications", *USENIX SANS III Conference Proceedings*, 1994.

Appendix A: A Sample Configuration File

Here is a sample of what a configuration file for "runprog" could look like:

```
!
! this file is a sample of the things
! necessary and possible in a "runprog" configuration
!
! 12 Aug 1996   JDBell   (jbell14@ford.com)
!
! lines beginning with "!" are comments
! whitespace is ignored
! first, a variable assignment (this is the usage log file)
ACCTHOST = /ford/lib/runprog/client-log
! a couple of program (class) definition
defprogram Generic {
    flags = none
    maxUsers = unlimited
    maxNetworkUsers = unlimited
}
! the "System" class inherits from the "Generic" class,
! with certain modifications
defprogram System: Generic {
    class = System
    flags = status
    acctHost = %ACCTHOST
}
! now, a directory configuration (which happens to be
! architecturally mapped)
GNUDIR = /ford/server/loc/GNU
directory GNU {
    link = %GNUDIR
    path = /ford/red.pto/loc/GNU/%arch
}
! next, several programs which are found in that
! directory
program gmake: System {
    path = %GNUDIR/bin/make
}
program gcc: System {
    environment = ( CC=gcc LIBSDIR=%GNUDIR/lib )
    path = %GNUDIR/bin/gcc
}
program gdb: System {
    path = %GNUDIR/bin/gdb
}
! finally, a single program which is architecturally mapped,
! showing off some of the configuration flags
EMULATORSDIR = /ford/server/loc/emulators
directory emulators_dir {
    link = %EMULATORSDIR
    path = /ford/black.pto/loc/emulators
}
```



```

program kermit: Generic {
    flags = SET_UID
    uid = uucp
    maxUsers = 2
    path = %EMULATORSDIR/kermit/kermit.%arch
}

!
! now, the "boilerplate" which makes the include file come out...
!
<< '
/* File: sample-configuration.h
** Template Date: June 2, 1992; August 30, 1994
** File Creation Date: %date
**
** created by %user on %host
**
*/

count := 0
with each program
{
    if userList then
    {
        << 'char *__string_list_%(count)[] = { %userList, NULL }; \n'
        count := count + 1
    }
    if machineList then
    {
        << 'char *__string_list_%(count)[] = { %machineList, NULL }; \n'
        count := count + 1
    }
    if environment then
    {
        << 'char *__string_list_%(count)[] = { %environment, NULL }; \n'
        count := count + 1
    }
}
if count > 0 then
{
    << '\n\n'
    count := 0
}
<< 'ProgramType programList[] =
{
total := 0
index := 0
with each program
{
    total := total + 1
    << ' { '
    << '"%(name)", '
    << '"%(path)", '
    << '%#(name), '
    << '%(flags), '

```

```

    << '%(uid), '
    << '%"class)", '
    << '%(maxUsers), '
    << '%(maxNetworkUsers), '
    if userList then
    {
        << '__string_list_%(count), '
        count := count + 1
    }
    else
    {
        << 'NULL, '
    }
    if machineList then
    {
        << '__string_list_%(count), '
        count := count + 1
    }
    else
    {
        << 'NULL, '
    }
    if environment then
    {
        << '__string_list_%(count), '
        count := count + 1
    }
    else
    {
        << 'NULL, '
    }
    << '%"lockHost), '
    << '%"acctHost), '
    << '%(index)'
    << '},\n'

    index := index + 1
}
<< '\n ENDOFPROGRAMS
};

int totalProgramCount = %(total);
total := 0
<< '
DirectoryType directoryList[] =
{
with each directory
{
    total := total + 1
    << ' { '
    << '%"name)", '
    << '%"link)", '
    << '%"path)", '
    << '%!(link), '
    << '%!(path), '
    << '%(flags), '
    << '%"prepare), '
    << '%"install), '

```

```

        << '%'(cleanup) '
        << '},\n'
    }
    << '\n  ENDOFDIRECTORIES
    };

    int totalDirectoryCount = %(total);

```

And here is the generated include file (which is compiled with the "runprog" sources):

```

/* File: sample-configuration.h
** Template Date: June 2, 1992; August 30, 1994
** File Creation Date: 6:21 pm Tuesday, August 13, 1996
**
** created by jbell14 on jdb500
**
*/

char *__string_list_0[] = { "CC=gcc","LIBSDIR=/ford/server/loc/GNU/lib", NULL };
ProgramType programList[] =
{
    { "gcc", "/ford/server/loc/GNU/bin/gcc", 0x67636300, STATUS, 0, "System", -1,
      -1, NULL, NULL, __string_list_0, NULL, "/ford/lib/runprog/client-log", 0},
    { "gdb", "/ford/server/loc/GNU/bin/gdb", 0x67646200, STATUS, 0, "System", -1,
      -1, NULL, NULL, NULL, NULL, "/ford/lib/runprog/client-log", 1},
    { "gmake", "/ford/server/loc/GNU/bin/make", 0x676d616b, STATUS, 0, "System",
      -1, -1, NULL, NULL, NULL, NULL, "/ford/lib/runprog/client-log", 2},
    { "kermit", "/ford/server/loc/emulators/kermit/kermit.solaris", 0x6b65726d,
      SET_UID, 5, "", 2, -1, NULL, NULL, NULL, NULL, NULL, 3},
    ENDOFPROGRAMS
};

int totalProgramCount = 4;

DirectoryType directoryList[] =
{
    { "GNU", "/ford/server/loc/GNU", "/ford/red.pto/loc/GNU/solaris", 20, 29,
      0x0, NULL, NULL, NULL },
    { "emulators_dir", "/ford/server/loc/emulators",
      "/ford/black.pto/loc/emulators", 26, 29, 0x0, NULL, NULL, NULL },
    ENDOFDIRECTORIES
};

int totalDirectoryCount = 2;

```


Automating the Administration of Heterogeneous LANs

Michael Fisk – New Mexico Institute of Mining and Technology

ABSTRACT

The areas of machine configuration and software package installation and maintenance have been frequent areas of work in recent years. This paper describes a hybrid system developed to address both problems and more. The resulting system is designed to reduce the complexity of the administration of a large network of computers down to that of the administration of a few heterogeneous systems.

In particular, this system allows machines to be maintained without ever having to manually change files on their disks. Systems can also be upgraded, installed from scratch, or recovered with a minimum of effort. The system described is designed to be extremely general and applicable to virtually all versions of UNIX and UNIX-like operating systems.

Design Goals

The development of this system was driven by two primary goals. First, consistency should be maintained between machines whenever possible. Machines should have the same configurations and same software unless specifically designed otherwise. At a configuration level, this means assuring that similar machines use the same nameservers, run the same daemons, and generally operate identically. At the software level, it means that the same version of a package should be installed the same way on all architectures and all machines whenever possible.

Second, the process of installing and upgrading machines should be automated as much as possible. New hardware, operating systems, and configuration changes should be easily assimilated into the configuration of the local network. Upgrading to a new operating system should not require scheduling a large block of down time and a correspondingly large amount of human labor. Similarly, it should be easy to install any single new machine so that it matches all similar machines on the network.

More specifically, the resulting system should simplify the following tasks:

- Installing and configuring new machines without forgetting any of the myriad of small details that make a machine operate as expected with a network.
- Making small configuration changes to all current machines.
- Performing major OS upgrades to all current machines.
- Upgrading services such as mail or NNTP.
- Migrating services between servers. As we upgrade our servers and separate services onto discreet "toasters," moving a service, such as mail or NNTP, to a new machine was a painstaking process involving too much trial

and error. Making sure everything was configured correctly on the new server and removing all superfluous things from the old server was too labor intensive.

- Maintaining a server-based `/usr/local` filesystem for multiple architectures. As with most sites, our clients mount a `/usr/local` filesystem from a server. Due to the extremely diverse needs of our user community, our `/usr/local` is quite large and exists for four incompatible flavors of UNIX on different hardware platforms. Installing a package for all architectures so that it behaved identically on each was too time consuming.
- Migrating packages from a server-based `/usr/local` filesystem to local disk on machines with larger disks.
- Maintaining specialized packages on individual machines. There was a growing need to be able to install and maintain machines in peoples' offices with special software that is not available over the network to all users.

In addition, we sensed the opportunity to use the same infrastructure to provide the following functions that had also been identified as tasks that were overly troublesome or redundant.

- Maintain legal DNS files. These files seemed to be constantly incorrect and often contained illegal syntax from some of our less experienced system and network administrators.
- Maintain correct `/etc/ethers` file. This file is routinely neglected, but used for some network management functionality.

Previous Work

OMNICONF [Imazu] isolates configuration files from operating system files, and provides mechanisms for storing and then restoring the configuration files. It

does not, however, aid in installing the operating system itself. It is also not designed for changes to be made to a central template and automatically appear on each workstation. Instead, changes are supposed to be made on a sample workstation and the differences stored for application to other machines. *OMNICONF* also seems to lack an easy way of sharing some, but not all, of a configuration between similar, but not identical machines.

GeNUAdmin [Harlander] provides extensive mechanisms for maintaining configuration files on machines, but is not designed to maintain whole installations of large software packages. *Config* [Rouillard1, Eirich] is better suited to maintaining both configuration files and whole software packages, but depends on a potentially complex *rdist* file and a primitive database for describing machines so that they may be grouped for *rdist*. In addition, the use of *rdist* implies a push rather than a pull mechanism. This can cause problems when changes are pushed while machines are down.

Sasify [Shaddock] uses a set of action scripts to perform arbitrary operations on machines. The appropriate action scripts are run using hostclasses representing logical groupings of machines. *Sasify* also uses a more attractive pull paradigm at reboot to insure that changes get propagated. Unfortunately, *Sasify* lacks the infrastructure needed for managing large sets of software packages for multiple architectures.

lcfg [Anderson] is designed to solve many of the same problems that this system addresses, and therefore provides much of the same functionality. It is unclear, however, how well *lcfg* handles the requirement for different software to be installed on different machines. *lcfg* makes extensive use of dynamic configuration at boot-time. It is therefore intentionally less sophisticated in maintaining static configuration information.

What is desired is a system that uses *Depot*-style [Rouillard2] mechanisms for storing and arranging software, but that also provides ways of maintaining machine-specific configuration files and software subsets. As discussed below, the system should also require clients to periodically pull new changes from a server rather than depending on a server being able to push them.

New Concepts

Static vs. Dynamic

As discussed by Anderson [Anderson], there is a trade-off between statically configuring machines by putting their configurations on local disk and dynamically configuring machines by having them query a server at boot time.

Our site has experimented with both types of operations. Our experience has led us to believe that for reliability and performance, all configuration

information should be stored locally on the machine. This allows machines to boot without the presence of the server. While our main server is generally up, we have been working hard to make our clients as independent as possible. Not only does this reduce load on the server, but in cases where large numbers of machines reboot after power outages or similar disasters, it allows machines to boot correctly even if the server is not up yet.

Changes to configuration information should be made on a central server and then propagated to the client. This is most robustly done by pushing information to all reachable clients **and** by having all clients check the servers periodically or at boot-time. This guarantees that machines that are down or unreachable won't miss changes, but also allows for the immediate propagation of changes without intensive, frequent queries of the server.

Representation of Configurations

Many previous systems for machine configurations are based on performing a series of sequential operations on a machine [Shaddock, Eirich]. All operations must be saved and applied to machines to insure proper configuration. This leads to the problem of determining whether certain actions have been superseded by others. It was the conclusion of the author that the configuration of any typical UNIX workstation can be represented solely on the contents of its filesystems. Therefore, the configuration process need not contain a series of sequential operations that need to be run on the machine. Instead, a machine's configuration can be completely represented as the union of the correct set of files.

Therefore, the system is built around a set of separate packages. Each package is the files necessary for some logically discreet functionality. For instance, *X11R6*, *Emacs*, and *INN* are all typical packages. In order to preserve generality in the system, the vanilla operating system, as installed from original media, is stored as a package. When a machine is installed from scratch, the operating system package is installed first. Site-specific configuration packages are then loaded on top of that.

Previous work [Rouillard1, Eirich] has been done based on enumerating files and directories that should be copied or installed on machines or groups of machines. Packages offer a more implicit way of doing this since these files are often stored together for convenience anyway. Using packages, directory layout and organization can be represented in their most natural states, directories.

A central database is maintained to define which packages should be installed on any given machine. Hostclass [Fletcher] definitions are awkward for adding new configurations. It is harder to locate all classes that a given machine belongs to and to duplicate those for a new machine. In addition, classes are then usually associated with actions or groups of files

in a separate database. Because of this, it is difficult to evaluate the effect of adding a machine to a given class.

For these reasons, the database is designed to make use of inheritance hierarchies. A machine is defined as an object. It can contain arbitrary variable/value pairs as well as a list of other objects to inherit variable/value pairs from. Objects can be defined for any logical group of machines that share some configuration information. This structure allows a machine to have a large amount of information associated with it, while still allowing short, but flexible, definitions for machines. It also makes cloning the configuration of a new machine off of an existing one easy. Additionally, exceptions to standard configurations are allowed for.

Creating Packages

The operating system package is generally made by installing a machine from scratch from the vendor's CD. Once the operating system is installed and configured to the point where it can talk to the network, the filesystem is dumped into a package directory on the installation server.

Our site runs Linux on over 100 PC's. A few of these still have very small hard disks. For this reason, we have two different versions of the Slackware package. One version includes virtually all packages in Slackware. The other is a stripped-down version that is sufficient to boot a machine, but will fit on a small disk.

Our site also has a couple of DEC Alpha servers. Since DEC releases an operating system update as often as we install a machine, we have not placed the OS in a package. Instead, we install the machine from CD as is usually done and then just execute a partial installation to configure the machine. Since we will typically not install this same operating system version again, we do not bother copying the OS into a package on the installation server as we normally would.

Structure

The system is composed of three main components, described below.

Machine Database

The machine database, or *Machdb* as it is called, defines all machine-specific variables. These range from IP address to nameserver to a list of packages to install. The database is currently implemented as a text file defining any number of objects. Each object can contain any number of variable/value pairs and can inherit from any number of other objects, recursively. This technique allows a definition for a machine to consist of only a few unique values such as IP and MAC address. All other values can be inherited from other objects.

The database allows set operations on variables in order to support combinations of different software sets as well as exceptions to standard configuration. For example, assume that all client machines are defined to inherit from an object by the name of `standard-client` that defines

```
PACKAGES=amd ypclient X11R6 bash emacs
```

Now assume that TeX is not a commonly used package at the site, but that it is required on some machines. The object for a client may inherit from `standard-client` to get the default list and also define `PACKAGES+=tex`. The `+=` operator will append rather than destructively setting the value for that variable. In addition, that client can also define `PACKAGES-=ypclient` to prevent it from getting the YP configuration files even though `ypclient` is listed in the `PACKAGES` definition in `standard-client`.

The database is queried via the *machdb* program which is a Perl script that parses the file, expands inheritances, and produces responses in various formats. One such format prints out all hosts in DNS record format suitable for direct use by BIND. There is also an option to check all machine objects for the existence of certain required fields. A set of sample database entries and query results are included as an appendix.

Packagelink

Packagelink is a Perl program that takes hierarchically organized package directories for multiple architectures and builds a single target directory for users to access. It was originally designed to be run on a fileserver to build a `/usr/local` filesystem containing hard links to all appropriate files for that architecture. In the last two years, it has been expanded to build mirror filesystems by building the target filesystem on a different machine than the packages reside on and copying files instead of linking them. *Packagelink* is designed to be extremely flexible and can build not only `/usr/local`, but any filesystem. This method of operation is used to build `/` on a machine from a set of packages on the installation server.

Packagelink is designed to operate on running systems. For that reason, it is careful to not delete or change existing files until it is sure that the current version is different from the version to be installed. Therefore, it can be run on a well-configured system and will make no unnecessary changes that would hinder other processes.

While copying files, if *Packagelink* encounters any files ending in a particular suffix (`.plprocess` by default), it will run those files through an internal preprocessor. Any strings of the form `'##VARIABLE##'` will be replaced with the corresponding value specified by a `'-DVARIABLE="VALUE"'` command line option. Conveniently, *machdb* can

output a list of `-D` arguments for all variables defined for a machine. This preprocessing serves as a substitute for having a script generate machine-specific configuration files or for needing a separate package for every machine.

The `###` syntax is also used since `#` is the traditional comment character for UNIX configuration files. Thus, a single comprehensive `inetd.conf` (or other configuration file) can be created for an entire group of machines. Services that will only run on some machines can be put in lines beginning with a `###SERVICENAME###` directive. This will remain in the file unless `'-DSERVICENAME=""'` is specified as an argument to *Packagelink*. If so, the comment will be removed and the service activated.

Most services, however, can be optionally installed using a System V style directory of start scripts where each service has its own script. Script names are alphanumerically ordered so that they are started in the correct order. Each script is stored in the package that it starts. Therefore, if a package is not installed, the script will not be placed on the machine and will not be run. BSD-style systems can have this functionality added by appending a simple *for* loop to the end of `rc.local`. This eliminates the need to maintain different `rc.local` scripts for each machine.

Packagelink has used various algorithms for determining how to combine packages into the proper filesystem. Currently it scans all packages and builds a database in memory of every file in every package. This database is used to resolve conflicts between different packages. Different packages can be given different priorities so that if a file exists in more than one package, the version in the package of highest priority will be installed. If duplicate directories exist, *Packagelink* can be configured to always merge them or to only merge packages of the same priority.

The database built by scanning packages is dumped into a file for subsequent executions of *Packagelink*. Using this saved database, future duplicates can be resolved without scanning all previously installed packages.

A *Packageunlink* program is also planned. This program will remove all files for a specified package and replace them with any duplicate files that had previously been overridden because they were in packages of lower priority.

A *Packagecheck* program will also be written to verify that all files on a machine came from a package and are correct. This will provide some of the functionality of *Tripwire* [Kim].

Installation Program

The main program, affectionately called *gutinteg*, is also written in Perl. In its simpler mode of operation, the default, the program configures an already running machine by performing a partial install. This

is usually done by running the script by hand or from a periodic cron job. The program probes a machine for its MAC addresses using `ifconfig` and `dmesg`. The *Machddb* entry for that MAC address is then loaded. *Packagelink* is then called to install software on the machine. Using `/` as the target filesystem, *Packagelink* copies all files onto the machine. *Gutinteg* then reinstalls the boot block or boot loader on the root partition.

For complete installs, the machine is booted from an alternate device. External SCSI drives, boot floppies, and network filesystems have all been used. The install scripts is added to the end of the normal boot sequence. In addition to the normal steps of a partial install, in this mode, *gutinteg* makes use of the `PARTITION` variable in the machine database to partition and format all disks. A sample value for a Solaris machine is:

```
c0t3d0: (a 500 ufs /) (b 500 swap /tmp)
c0t1d0: (h 1000 ufs /home data)
```

This value specifies a 400 cylinder partition for `/`, and a 500 cylinder partition for swap (to be mounted as `/tmp` on machines that support Sun's `tmpfs` [Snyder]). A 1000 cylinder partition for `/home` is created on a second disk. The `'data'` flag means that this partition will not be formatted during installation.

The partition letters are abstract. On operating systems with numbered partitions, they will automatically be converted to 0-6 or 1-7 where appropriate. On operating systems that traditionally have a 3rd partition containing the entire disk, one will be automatically created. On Linux machines, extended partitions and other DOS-isms are also handled automatically.

After partitioning and formatting disks, *Gutinteg* installs the appropriate operating system package(s), and proceeds with the steps of a partial installation. Finally, operating system dependent functions such as configuring `/dev` and `/devices` are performed and an `fstab` or equivalent file is produced from the `PARTITION` list.

DOS

Our group is also responsible for maintaining a Novell network of DOS machines. We decided to require that Linux be installed on all of these PCs so that we can use UNIX tools to maintain the local DOS partitions. The machines automatically reboot every night to Linux, where the DOS partition is mounted as `/dos`. Currently, a cron job runs just after that to fix any changes that users may have made to the local disk during the day. This functionality is being replaced by the new system which will maintain `/dos` as it maintains every other directory.

Usage Experience

A previous version of the system was used to upgrade 26 Sparc workstations to SunOS 4.1.4 in the

fall of 1995. We also used this opportunity to have the system repartition all of the disks at the same time. A single, external, boot disk was used to boot each machine for installation. The process was so trivial, that our User Consultants (who are not trained in UNIX system administration) upgraded all of the workstations without assistance from any of the system administrators. While the process could have been accelerated by remotely booting several of the machines off the network at once, this method allowed us to keep all but one of the workstations up at all times for use.

The current version has been used for all new Linux workstations purchased or evaluated in the last few months and all Linux and Sun machines on our test network. A complete reinstall of over 100 Linux machines was performed this summer by one programmer during a weekend.

Complete installations can take anywhere from 30 min to over an hour depending on the size of the system, network traffic, etc. The main bottleneck for reinstalling a large number of systems is the ability to do so simultaneously. Ramdisks on Linux machines and network root partitions that can be used by multiple machines simultaneously have been key to allowing us this capability.

In addition, we are currently working to populate the installation server with packages for all of the key services on our main servers. Once this has been completed, the system will be used to reinstall all of our servers.

Once all of our machines have been reinstalled using the system, we will abandon all other methods of maintaining local machines, and switch to running the partial installation script automatically on a regular basis.

Conclusions

The system described in this paper has allowed our system administration staff to stop maintaining individual machines and instead maintain a single large template of packages and configuration files. Since the files are still stored in traditional directory hierarchies, the learning-curve for maintaining the new system is minimal. As a result, maintaining a large network of machines is becoming less related to the number of machines, and instead proportional only to the number of different architectures, operating systems, and software packages.

Availability

The three tools comprising this system, *gutinteg*, *machdb*, and *packagelink* are all freely available Perl programs. They are, however, not yet designed for public consumption. Anxious readers are welcome to contact the author for information on how to retrieve current versions. Note that they will largely be unsupported software, but I am always willing to work with

other users and, if necessary, modify the programs to make them useful to a wider audience.

Acknowledgments

The author would like to thank K. Scott Rowe for his continuing work with and experience using this system, Tony Heaton for his assistance in testing it, and James Robnett for allowing the author to pursue good ideas when he found them.

Author Information

Michael Fisk developed this system while employed by the New Mexico Institute of Mining and Technology's Computer Center. He has since completed his B.S. in Computer Science from that university and is currently a staff member at Los Alamos National Laboratory in the Network Engineering Group of the Computing, Information, and Communications Division. His current work includes introducing the topic of this paper to system administrators at the Lab, prototyping protected open networks, and researching other areas of internetworking infrastructure. He can be reached by e-mail at mfisk@lanl.gov or by postal mail to: Michael Fisk; CIC-5/MS B255; Los Alamos National Laboratory; Los Alamos, NM 87545.

References

- [Anderson] Anderson, Paul, "Towards a High-Level Machine Configuration System", *LISA VIII Proceedings*, 1994.
- [Eirich] Eirich, Thomas, "Beam: A Tool for Flexible Software Update", *LISA VIII Proceedings*, 1994.
- [Fletcher] Fletcher, Mark, "doit: A Network Software Management Tool", *LISA VI Proceedings*, 1992.
- [Harlander] Harlander, Dr. Magnus, "Central System Administration in a Heterogeneous Unix Environment: GeNUAdmin", *LISA VIII Proceedings*, 1994.
- [Imazu] Imazu Hideyo, "OMNICONF - Making OS Upgrades and Disk Crash Recovery Easier", *LISA VIII Proceedings*, 1994.
- [Kim] Kim, Gene H. and Spafford, Eugene H., "Experiences with tripwire: Using integrity checkers for intrusion detection", *SANS III Proceedings*, 1994.
- [Rouillard1] Rouillard, John P. and Martin, Richard B., "Config: A Mechanism for Installing and Tracking System Configurations", *LISA VIII Proceedings*, 1994.
- [Rouillard2] Rouillard, John P. and Martin, Richard B., "Depot-Lite: A Mechanism for Managing Software", *LISA VIII Proceedings*, 1994.
- [Shaddock] Shaddock, Michael E. and Mitchell, Michael C. and Harrison, Helen E., "How to Upgrade 1500 Workstations on Saturday, and Still Have Time to Mow the Yard on Sunday", *LISA IX Proceedings*, 1995.

[Snyder] Snyder, Peter *tmpfs: A Virtual Memory File System*, <http://www.sun.ca/white-papers/tmpfs.html>, Sun Microsystems, Inc.

Appendix: Sample Database Definitions

```
underdog: dot3 Standard486 speare20
          IP=129.138.3.162
          MAC=00:40:33:2d:08:3f

dot3:     classC tcct.nmt.edu
          GATEWAY=129.138.3.1
          BROADCAST=129.138.3.255
          NETWORK=129.138.3.0

classC:
          NETMASK=255.255.255.0

tcct.nmt.edu:
          NAMESERVER=129.138.3.220
          NAMESERVER2=129.138.4.216
          DOMAIN=tcct.nmt.edu
          YPSERVER=teal

speare20:
          LOCATION=speare20
          PRINTER=speare16

Standard486: i486 mach32 at1500

at1500:
          NIC=ne1500t
          PACKAGES+=at1500

i486:     full-linux
          CPU=i486
          ARCH=i486

mach32:
          VIDEOCARD=mach32
          PACKAGES+=mach32

full-linux: small-linux
          OSREV=Slakware3.0
          PACKAGES+=xfig transfig

small-linux: base-linux
          OSREV=small-Slakware3.0
          PACKAGES+=nntpcient

base-linux: dos
          OS=linux
          OSREV=Slakware3.0
          ROOTFS=hda2

dos:
          PACKAGES+=dos
```

```
q-bert.NIC="ne1500t"
q-bert.VIDEOCARD="mach32"
q-bert.CPU="i486"
q-bert.ARCH="i486"

q-bert.GATEWAY="129.138.3.1"
q-bert.BROADCAST="129.138.3.255"
q-bert.NETWORK="129.138.3.0"
q-bert.NAMESERVER="129.138.3.220"
q-bert.NAMESERVER2="129.138.4.216"
q-bert.DOMAIN="tcct.nmt.edu"
q-bert.YPSERVER="teal"
q-bert.NETMASK="255.255.255.0"
q-bert.OS="linux"
q-bert.ROOTFS="hda2"
```

Sample Database Query Result

```
q-bert.IP="129.138.3.162"
q-bert.MAC="00:40:33:2d:08:3f"
q-bert.PARTITION="hda: (a 51 msdos) (b 421 ext2) (c 53 swap)"
q-bert.PACKAGES="xfig transfig nntpcient at1500 mach32"
q-bert.LOCATION="speare20"
q-bert.PRINTER="speare16"
q-bert.OSREV="Slakware3.0"
```

PC Administration Tools: Using Linux to Manage Personal Computers

Jim Trocki – American Cyanamid Company

ABSTRACT

Personal computers in a networked environment can provide users with access to a broad set of distributed resources. Unfortunately, the management overhead of maintaining PC clients can become overwhelming, especially with a large installed base. Popular PC operating systems do not provide system administrators with a set of efficient and flexible management tools that can take advantage of a networked environment. UNIX system administrators are accustomed to having such tools at their disposal to handle common administration tasks, such as software upgrades, initial machine installation, networked file transfer, and remote backup.

This paper describes the PC Administration (PCADM) tools developed to provide PCs with a UNIX environment and robust tool set for client administration purposes, without installing supporting software on individual clients. Linux, custom scripts and libraries, MD5 signatures, and freely available software including Perl [Schwartz], Bash, and SAMBA are used to accomplish this task – all made accessible from a single floppy disk.

Motivation

Managing PCs is time-consuming and cumbersome, partially due to how desktop PC software has evolved from a stand-alone environment into a larger, networked setting. In order to minimize time spent managing clients, support personnel need the ability to install and configure software on multiple machines simultaneously. This is both a technical and a logistic problem; most PC software installation programs are highly interactive, which limits the possibility of simultaneous installations for each PC support person, since physical attention is required for each client. The PCADM tools were designed to minimize, if not eliminate, interaction for software installations, and to provide a high level of remote accessibility to perform these tasks.

Not only is software installation and upgrading less than convenient, but the accessibility of the data stored on the local client is extremely limited. This complicates transferring data to alternate machines for analysis, an important need in a scientific R & D environment. Backup of data on a client PC directly to tape or other medium on a UNIX workstation is extremely difficult with tools supplied with popular PC operating systems. The PCADM tools provide this type of accessibility without expensive and proprietary PC software.

Design Goals

UNIX supplies an excellent environment for system administration compared to typical PC operating systems. The “tool set” approach is far more valuable for computer administration purposes than the “one big tool”. There should be a set of extensible tools which can be easily applied to new situations which appear in PC administration. These tools should fit

into the shell scripting paradigm in order to leverage from other shell-oriented tools.

User interaction should be limited whenever possible, and the input of redundant information should not be necessary. When performing maintenance tasks on a large number of clients, both speed and accuracy are important. Jobs performed in non-interactive mode complete faster than those which require persistent user interaction. If interaction is required at all, it should happen as near as possible to the initial invocation of a utility.

Minimal or no software should have to be installed on the client. If client software installation is a requirement, then it becomes an unwanted part of the overhead of administering PCs. This is what we are trying to reduce.

When possible, network configuration (IP address, etc.) should be gathered from already installed PC networking software, e.g., Sun PC-NFS, DEC Pathworks, and Microsoft Windows 95.

A Functional Linux System from a Floppy

Linux was chosen as the base system because of its broad PC hardware support, the support for the DOS FAT filesystem, and its availability.

All Linux installations provide bootable floppy disks which contain a kernel, libraries, utilities, and scripts used to perform the installation. The floppy normally contains a Minix filesystem which is loaded into memory by the ramdisk driver, and then mounted as root. The PCADM boot floppy is based on this technique for booting. Network drivers for several adapter models used on site are compiled into the kernel. In some cases, there are boot disks including alternate drivers for less common ethernet cards. This

method has the advantage of portability between machines, and does not require software to be installed on the client disk.

Obviously, there is little room for a complete system on a single floppy, so the first priority during the boot process is to establish IP networking in order to NFS mount /usr from a file server, which will hold all additional non boot-specific software. Software is then limited to whatever the administrator wishes to make available. Shell utilities such as sort, find, expr, grep, head, etc. are in the implementation, in addition to a vi clone and networking utilities like telnet, ftp, ping, and rsh.

The floppy is constructed to hold the boot loader, a Minix filesystem containing a Linux 1.2.13 kernel, libc and libm, ifconfig, route, a reduced shell, Perl 4.036, and custom scripts. This will be referred to as the PCADM boot disk.

LILO As The Boot Loader

The LILO [Almesberger, 1995] boot loader serves the purpose of loading a kernel image from the floppy. It also allows the user to supply kernel parameters (usually for drivers), but any parameters which are not recognized by the kernel are set as environment variables. The PCADM boot scripts take advantage of this feature, and IP configuration hints can be passed via LILO parameters. Other user-supplied parameters are used to enable the SAMBA daemon for SMB (LanManager) protocol exporting of the local disk, to prevent IP information from being acquired from locally-installed software, and more.

The following example will tell the boot scripts to use the IP address of "10.0.0.1", set the hostname to "pc001", export the "/dos" directory using the SMB protocol, to ignore the locally installed TCP/IP configuration, and to give the user one second to make changes to the network configuration settings:

```
LILO: param ip=10.0.0.1 hn=pc001 \
      samba=1 nogrep=1 wait=1
```

Currently recognized options are:

ip	IP address
hn	Hostname
nm	Netmask
na	Network address
gw	Default router
ns	Nameserver
dm	Domain name
samba	Start SMB server exporting /dos if nonzero
nogrep	override local TCP/IP software configuration if nonzero
wait	give user time to change configuration if nonzero

Default configurations are stored both by LILO as kernel parameters, and also in a "defaults" file on the boot floppy.

Memory

The PCADM disk will work properly on a machine with eight or more megabytes of RAM. Mileage may vary with less memory, and less is definitely not recommended. No attempt is made to establish a swap partition or a swap file, so all operations to be done should be able to fit in physical RAM. For our purposes, eight megs has been suitable. Four megs will not work. Since nearly all networked PCs on our site have at least eight megs of RAM, this does not hinder the use of the PCADM disk.

Boot Scripts

The boot scripts perform the function of acquiring the network configuration for the current host from the user, from boot loader hints, or automatically from DOS TCP/IP software which is already installed. Because this task is easier accomplished in Perl rather than sh, the Perl interpreter must reside on the boot floppy.

The boot scripts mount the local DOS FAT (the native DOS filesystem) boot partition on to the root filesystem (which is located in the ramdisk) as /dos.

If no IP parameters are supplied via the boot loader, scripts attempt to identify locally installed PC networking software, such as Sun PC-NFS or DEC Pathworks. Based on what is found, the network configuration (IP address, netmask, default gateway, and nameserver) is extracted from the respective configuration files for the product (/nfs/network.bat and /nfs/hosts for PC-NFS, /pw/cfg0001.tpl for Pathworks), and used for the host.

If none of the above suffices, the scripts prompt the user to confirm the configuration, and parameters may be interactively changed. Since we favor minimal interaction, a small timeout can be set via a LILO parameter, so that it is possible to insert the floppy into several machines and not need to attend to them, other than powering them on, and all further operations can be done from a remote location via Telnet.

The final stage in booting, once networking has been configured, is to mount /usr from a server. The rest of the necessary software is located on the server, including software installation packages, miscellaneous binaries, and daemons (telnetd, ftpd, rshd, etc.). /bin/sh is removed from the ramdisk filesystem (which was previously a minimal shell), and a symbolic link is made to /usr/bin/bash.

Remote Accessibility

Once the system has booted and /usr is mounted from the server, inetd is run. It is then possible to telnet, FTP, or rsh to the client PC, allowing full operation of administration tasks from any location via the network. Standard, open protocol, non-proprietary utilities are used to connect to the client to perform maintenance functions. Nearly any host which

supports TCP/IP will have Telnet, FTP, or rsh client utilities installed, so the administrator need not install the "other half" of remote administration software packages on to the workstations where the maintenance is to be done from. This is the functionality that all current PC networking packages lack – especially the capability of executing commands on the client host.

Accounts reside only in `/etc/passwd` on the floppy. The password-protected "root" account and a password-less "user" account allow login. Only root may write to the local filesystem. We are not overly concerned with the security of the PC, since physical security of the typical PC is normally non-existent, and anything could be done to the PC with or without the use of the PCADM disk. DOS or Windows provide no effective security mechanisms, anyway. Any software which attempts to limit access to the PC can be overridden by booting a floppy disk, and a password-protected BIOS is ineffective without physical security.

The PCADM effectively allows anyone with a PC to be root on that machine, and it does make it easier to exploit certain vulnerabilities in the servers. NFS security is one vulnerability that is easy to exploit, as is rsh (with the appropriate `.rhosts`). Also, various denial-of-service attacks are possible because the local IP address is completely configurable. Password sniffing is also possible with `tcpdump`. However, these vulnerabilities exist with or without the PCADM floppy, and the system administrator should already be taking appropriate precautions.

PC Software Installation

Since Linux is able to read and write to a DOS FAT filesystem, it enables us to install and configure software from an arbitrary vendor using the PCADM disk. The challenge then becomes mimicking the behavior of the vendor's installation code. Using Rivest's MD5 message digest algorithm [Rivest, 1992] and some other techniques similar to Spafford and Kim's Tripwire [Spafford et al, 1994], we are able to identify the exact results of the vendor's software installation program. This includes discovering files which are added to or removed from the client disk. The MD5 checksums identify exactly the files which were modified during the installation.

A MD5 checksum of each file on a PC with a bare minimum setup (DOS and Windows only) is recorded using the PCADM disk and the "makemd5db" script, which recursively traverses the local filesystem. Files which are suspect to modification (`CONFIG.SYS`, `AUTOEXEC.BAT`, all Windows INI files, etc.) are saved to a temporary location on a file server, and the machine is then rebooted with DOS. The vendor's software is installed using the provided DOS/Windows setup program, and PCADM is again booted. Another MD5 database is formed with

"makemd5db", and then a Perl script is used to compare the first and second checksum databases. The output is a list of files which were added, removed, or modified. Files which were changed (hopefully ASCII text files) are compared to the originals, which we previously preserved, using "diff -c", and the results of the software installation are identified.

Files which were added are collected from the client disk into a compressed TAR file, and moved to a location on the file server for later use. A MD5 database of checksums is also stored along with this image for individual file identification.

Scripts and Libraries

The PCADM disk provides a flexible environment with the proper tool set in order to complete the software installation. The next task is to generate a set of scripts which will reconstruct the installation on clients using the PCADM disk.

PC software installations can be separated into a few simple tasks: identifying dependencies, copying files to the client disk, and modifying several types of configuration files.

In the spirit of the "UNIX way", whenever possible and practical, we use existing tools which perform the tasks we need to, such as copying and modifying files. It would seem obvious to do such a thing, but in the world of PC software, each vendor seems to invent a wheel with a different diameter and a proprietary "axel interface", preventing it from being used on a someone else's (or particularly our own) cart. Only when existing utilities lack the functionality we need do we create our own tools. Kernighan and Pike's book [Kernighan, 1984] gives an excellent discussion of how the UNIX programming philosophy is based on the relationships between programs, and the method of using programs for building programs. This, combined with the expressive grammar of the shell, allows programmers to phrase their ideas in a flexible manner. Van Jacobson's keynote speech at USENIX '96 addressed this idea about what makes UNIX good, especially compared to popular PC operating systems.

Most dependencies for DOS or Windows software deal with particular versions of dynamic link libraries (DLLs), which can easily be identified using MD5 and compared to expected versions. Some PC software installation programs judge a DLL version by the date on the file, which is not unique enough criteria for identification. A priori knowledge about these types of dependencies from vendor documentation can be an advantage, however this will normally be discovered by trial and error.

The matter of copying files to the client disk is the simplest. However, it is not always adequate simply to copy files to the client, for fear of overwriting things which might already exist, and possibly losing valuable data. For this scenario, a utility is provided

which will rename files which already exist, and creates a DOS batch file on the fly. In case of some misfortune, this batch file can be executed under DOS, and things the files will be restored to their state before the installation occurred.

PC software installation programs will usually modify the same types of files. CONFIG.SYS and AUTOEXEC.BAT usually get modified, as do WIN.INI and SYSTEM.INI. Since these are common types of ASCII files, we would like a set of tools to make editing them simpler. In the case of AUTOEXEC.BAT and CONFIG.SYS, sed may or may not be the right choice of tool, depending on the type of modification. These files are free-form and have no apparent structure.

However, in the case of the standard Windows INI files, sed will not work without some trickery. Since these files have a somewhat rigid structure, it would be easiest to perform an operation like "add the entry BorderWidth=3 to the [Windows] section of WIN.INI". To make this type of phrasing possible, a Perl library needed to be developed to handle this type of file. Using the library, the code which accomplishes what this example wants is the following:

```
&change_ini ("/dos/windows/win.ini",
  "Windows & BorderWidth & 3") ||
  die "failure\n";
```

Other functions defined in the library will report the value of a variable in an INI file, since this type of operation is normal to the installation process. Utilities which will rename files to a unique name, perform logging functions, check free disk space, and write DOS batch files (among other things) are also included as part of the tool set for creating installation scripts.

Software Package Sets

Utilities are divided into sets specific for a given software package, e.g., Vendor A product, Vendor B product. Within a given software set there will be multiple scripts handling the logical stages of the installation of that package. A script set for a typical package will include the following:

- package.getdefs – "primes" the configuration by interpreting an existing configuration and supplying default hints to the following scripts. An example would be upgrading a version of networking software, and setting the parameters of the new version to be the same as the old version.
- package.getinfo – gets configuration information from the user. Here, interaction is minimized. The output of this stage will be a file holding "name=value" pairs, which will be interpreted by the configuration and setup scripts.
- package.setup – performs the operations of transferring the software files to the local

machine and modifies the local configuration to include the appropriate entries, but does not insert the user-entered configuration data.

- package.configure – modifies local configuration files to reflect the configuration options entered by the user in the "package.getinfo" script.
- package.install – a shell script driver which controls the overall process of an installation, from retrieving the defaults to configuring the package.

By separating the installation into somewhat discrete steps, we are able to make separate scripts that will configure or re-configure a machine which already has the package installed.

Drawbacks

The largest drawback is in the difficulty in identifying all dependencies by looking at installed files. Hopefully, adequate vendor documentation can aid in this process.

Proprietary and visually indecipherable data formats can be extremely frustrating. For what reason vendors choose to store small or medium sizes of configuration data in non-ASCII formatted files is beyond my own and several of my co-workers' understanding, but nevertheless, this is an all too popular practice. The Windows 95 registry is a notable offender, in addition to Windows 3.1 .GRP files and OLE registry.

The newly created installation scripts can be painstaking to debug, but careful initial preparation in the identification phase can minimize problems. The time saved (when amortized over a large number of PC installations) makes the debugging effort well worth it.

Timing Figures

By eliminating the user interaction, in some instances we can achieve some excellent speed increases. The example I cite is an installation of all standard software which is delivered to the client. This includes Microsoft Word, Excel, Lotus Freelance Graphics, Netscape, among other things.

Without the use of the PCADM utility, an installation would take approximately an hour and fifteen minutes or more to "get it right". This involved installing some software from floppy and some from distributions located on the network. The process is highly interactive, demanding constant attention from the person, so only one could be done at a time by a single person.

The same installation, when done with the PCADM utility, clocked in at less than ten minutes.

An additional scenario where the PCADM installation methods can be exploited is the setup and configuration of new PCs. Often an organization will

receive a volume of identical newer model computers which need the "standard issue" software installation and individual network configuration (IP address, default network filesystem mounts, etc.). In this situation, a single PC can be installed from a single, pre-configured "clone" machine in ten minutes or less. In addition, multiple machines can be installed simultaneously, stacked together without keyboards or monitors – just ethernet connections.

Data Transfer and Backup

The PCADM utility can also be used to turn a client PC into an NFS or LanManager server. This enables us to transfer data between "clients" instead of using temporary disk space on an intermediate server. This capability is useful when transferring software or data files from an older PC over to a newly purchased replacement machine. The conventional means for doing this was to backup the old machine to a portable quarter inch tape via the parallel port, and restore it to the new machine. This is an extremely slow and arduous task. With the PCADM disk, we have even left the old machine (without a keyboard or monitor) under users' desks, and the clients have used Windows File Manager to transfer only the files that they need, nearly eliminating the need for the intervention of the administrator.

The situation when locally stored data (word processing, spreadsheet, database files, etc.) must be transferred to another location can be problematic. Not always do the operating system files or applications on the local disk need to be moved, so it is desirable to identify the user data files from other files. However, a complication is that users often misplace files. A simple "find" command can find files with common extensions ("*.doc", "*.xls", etc.), but this technique alone is not enough to reliably select data. Users might possibly (and surprisingly often do) choose filenames that don't correspond with the conventions for the applications they use.

Because of the above problems, a filtering tool was developed that uses several criteria to identify user data. Filenames are first compared to regular expressions which attempt to match the normal filename extension conventions. If no match is found, then an additional comparison is used based on "magic numbers", which uses specific byte patterns in the beginning of the file to identify user data. Files which match any of the above criteria pass through the filter, which are usually fed as input to a copying utility such as "cpio -p".

Care must be taken in assembling the magic number list. If magic numbers are misinterpreted and omitted, it could possibly result in missing some important data during the transferral process. The method used for our purposes was to construct a list of the first five bytes of every file on one of our file servers (over 14,000 user data files). This list was

then grouped based on file extension, resulting in output similar to this:

```
dba52d00 7683 doc(7115 92.61%)
      tmp(123 1.60%) bak(100 1.30%)
      drl(901.17%)
```

The first column is the hexadecimal pattern for the first four bytes of the files, followed by the number of files with this pattern, and percentages of how many files from the second column contained that extension, i.e., over 92% of files starting with the pattern "dba52d00" ended with the extension ".doc". We know that the ".doc" files are from our word processing package, and after inspecting other high percentage occurrences of the "doc" extension, it is reasonable to use the string "dba52d00" as part of our magic number criteria.

File extension regular expressions are stored in a separate file to make it simple to add patterns, and has the following format:

```
\.do[ct]      Word processing file
\.xl[cs]      Spreadsheet file
\.pre,\.flw   Presentation pkg file
```

"Magic numbers" are stored in a file with the following format:

```
dba52d00      Word processing file
01020304      Spreadsheet file
```

This technique has so far yielded acceptable results.

Conclusions

The PCADM boot disk and tools have been useful to the administration tasks of our organization. They provide enough flexibility to customize software installations and reduce installation times. Many convenient operations simply are not possible without PCADM, particularly in the area of remote access.

Availability

Source code and images of the PCADM boot disks will be made publicly available. However, at the time of writing, the author is still looking for a facility from which the packages can be distributed. Volunteers with ample disk space on their FTP servers may contact the author at "trocki@pt.cyanamid.com", and further arrangements can be made.

Bibliography

- Rivest, R. "The MD5 Message Digest Algorithm," RFC 1321, 1992.
- Kernighan, B., Pike, R. "The UNIX Programming Environment", Prentice-Hall, Inc., 1984.
- Schwartz, R., Wall, L. "Programming Perl", O'Reilly and Associates, 1991.
- Spafford, E., Kim, G. "The Design and Implementation of Tripwire: A File System Integrity Checker", Dept. of Computer Sciences of Purdue University, 1994.

Almesberger, W. "Generic Boot Loader for Linux",
User's Guide, 1995.

Abstract Yourself With Modules

John L. Furlani – Sun Microsystems, Inc.
Peter W. Osel – Siemens Components, Inc.

ABSTRACT

Modules abstracts the activation of applications from the details of their installation. It provides a uniform interface for selecting applications and for applying the necessary changes to the environment.

Five years ago, the first paper on Modules was published [1]. Since then, Modules has been written in C, uses Tcl [2] as its extension language, has seen acceptance and use at a rich variety of sites and has acquired several features for supporting the management of hundreds of software packages across large and diverse intranets. With Modules' proven combination of features and flexibility, we believe it has the potential to become the preferred standard for software management and activation.

In this paper, we compare the Modules package with several systems that have appeared in the years since its introduction. We also present some real-world examples of how the Modules package is being applied. This paper covers some of the new features in the current implementation. Finally, we discuss how the Modules concept can be applied elsewhere, including the problem of loading and installing on-demand applets and applications.

Introduction

The problems surrounding software installation and software distribution for heterogeneous distributed networks continue to be an area of great interest to systems administrators and software developers alike. Because of this level of interest in recent years, a plethora of solutions are available for solving many of the problems with installing and distributing software for large networks. Beyond software installation and distribution is software activation and software management. We define *software activation* as the problems surrounding how a user learns about installed software as well as how the software, once installed, is enabled by the user. We define *software management* as the set of problems around maintaining an ever-growing, ever-revising array of software packages. Modules solves the problem of software activation and software management for the user and the systems administrator. In essence, Modules is about abstracting the user from traditional dependencies on the location, the architecture and the issues surrounding version migration for installed and distributed software packages.

In this paper, we present a few interesting case studies which exemplify how the Modules package can be and has been applied. We follow up the case studies with a section comparing Modules as it is implemented today with other packages and approaches that attempt to solve related problems. Once compared with other solutions, we describe in more detail the currently available implementation of the Modules concept done in Tcl and C. The final portion of the paper is dedicated to new features in future versions of the Modules package, to ideas for new

implementations of the Modules concept and to other areas where the concepts behind Modules can and should be applied – specifically applet deployment.

It is important at this point to separate the Modules package from the Modules concept. The *concept* behind Modules is grounded in applying object oriented techniques such as methods and data abstraction to the problems of software management and activation. The *Modules package* is a particular implementation of the Modules concept for UNIX systems using Tcl and C.

Modules Basics

Modules abstracts the activation of applications from the details of local software installation and distribution. It frees the user from clumsy manipulation of environment variables by providing high level methods for software activation.

If “object oriented” wasn’t an overloaded buzzword, we would be even more inclined to advertise Modules as the “object oriented” solution to software management and activation. In object oriented parlance, installed software packages are the objects. Modulefiles or some other application-specific database information contains the implementations for a set of methods, e.g., activate, deactivate, show information. The program behind the *module(1)* command implements a command-line user interface for triggering the execution of methods on the objects (deactivate *software1*, activate *software2*). The *module(1)* user interface does not differ between platforms or shells.

Dependencies and conflicts between applications and versions of an application can be configured into

the modulefiles. Modules also affords the user query abilities which include listing the available software packages, information about a particular package and package-specific help.

Case Studies

The Modules package is helping numerous sites of different sizes, ranging from a single workstation to large intranets with several hundreds or thousands of workstations, to manage their software. This section describes four deployments of the Modules package and exemplifies its flexibility.

Siemens AG

At the Semiconductor Division of Siemens AG the Modules package is used for managing more than 250 software packages on more than 800 Sun and HP workstations worldwide. The software packages include commercial, freely available and proprietary tools. It is not unusual to have many (sometimes up to a dozen) versions of a software package installed and in use at the same time. For example, each version of their CAD system uses a different set of software packages. The software and configuration information, including modulefiles, are mirrored to development sites in Germany (München and Düsseldorf), Austria (Villach), The United States (Cupertino, CA), and Singapore using OpenDist [9]. Within their environment *module(1)* is called more than six thousand times a day. They also ship parts of their CAD system to more than twenty external design contractors. At the contractors' sites the Modules package integrates smoothly into a multitude of different existing system and network setups.

All sites use the same modulefiles. Differences in the systems' configuration are handled by common library functions returning domain-specific and system-specific configuration information. Thus, all system configuration can be done by a central library, thereby decreasing the maintenance effort while improving documentation of differences between their computing sites.

Every software package activation is logged using the usage tracing feature of the Modules package. Knowing the usage of software on the systems at each site helps them efficiently phase out old or unused software. Usage tracing also enables them to inform affected users of a particular software package if a mission-critical bug is found. This is more efficient and effective than posting to a general information news system or broadcasting to all users. They have found that users bombarded by e-mail or repeated news about software packages they don't use end up ignoring all of the informational messages. They claim the timely and effective distribution of information to those affected saves thousands of dollars.

Sun Microsystems, Inc.

The Modules package originates from Furlani's work as a systems administrator for the North Carolina Development Center. Its use unified software management and activation for about seventy users on over one hundred workstations. Novice users found it simpler to learn about and activate new applications. Advanced users quickly embraced its wide range of capabilities and its immense flexibility.

Often, software developers must test against different releases of many different source code trees. The Modules package permits a team-organized mechanism for simple switching between daily builds, weekly builds, earlier releases, personal clones and other developer's clones.

Today, the Modules package is heavily utilized by the Solaris XIL Image and Video Library development team. The team shares a common directory of modulefiles specifying the location of common directories like daily, weekly and release builds. Each developer has his or her own directory of modulefiles augmenting or overriding the commonly available modulefiles. Often a developer isolates a bug or adds a new feature in his or her development tree that other developers want to test or try prior to integrating the changes into a master source tree. Switching to a new development tree occurs with a single *module(1)* command referencing the target developer's modulefile.

Also, some groups inside of Sun use the Modules package for the more commonly stated software activation and management features.

Cray Research, Inc.

The Modules package is released with Cray's Programming Environment 2.0 [10] product. Cray supplies modulefiles with the product for managing the dependencies between the various pieces within the Programming Environment as well as the cross-compiling capabilities of their compilers. Using the Modules package also allows local systems administrators an easy method for managing more than one version of the product at the same time. Cray also provides some tools for customers to create their own modulefiles as well as a specific modulefile for each version of the product.

Auburn University

The College of Engineering at Auburn University uses the Modules package to manage a large number of applications for many users. They use the Modules package as the foundation for a self-developed menu-driven application initialization and activation package called user-setup [4]. Since the introduction of user-setup Version 2 in May 1992, over 5,000 users have benefited from the easy menu-driven setup and manipulation of their environment afforded by user-setup.

The systems administrators found an immediate and dramatic reduction in their help-desk's work-load

after the introduction of user-setup Version 2. Some unexpected side-effects of introducing this Modules-based solution include simplified instructions for those describing how to access software applications, a reduction in requests to install new software packages, users easily locating and using a wider variety of software on their own and enabling the use of UNIX systems to non-UNIX users.

Related Tools

We divide the software distribution cycle into five distinct steps as follows:

- *Development* encompasses the creation of the bits comprising an application.
- *Packaging* focuses on the problem of getting the bits onto a distribution media (be that magnetic, optical or a network).
- *Installation* (the inverse of the packaging problem) is getting the bits off of the distribution media onto a local media in an organized fashion.
- *Integration* is making the bits available on the user's system.
- *Activation* includes how users learn about the availability of the application as well as how they enable the new application once it is integrated on their system.

Each stage of the software distribution cycle requires solving a different set of problems. Solutions and tools often cover more than a single step. In fact, they often imply or depend upon attributes of the neighboring steps.

Tools for software development, software packaging and software installation are beyond the scope of this paper. Although, it is important to note that Modules operates independently of the solutions used for packaging and installation. Modules aims to solve the final two steps of this software distribution cycle. The next two sub-sections focus on how Modules can be used to integrate and activate software in a distributed environment.

Integration

Conceptually, files on a user's system start at the root directory and live locally or remotely. Integration involves integrating the application into the filesystem hierarchy available to each user.

There are a number of solutions which provide the mechanics for organizing and distributing the application's bits to a large number of machines. Modules works independently of these software installation and integration solutions. But we have found that Modules better solves the distributed software management problem if Modules' object oriented characteristics are taken into consideration when choosing or utilizing an integration scheme.

Typically, solutions to the software integration problem provide policies dictating how software

packages are to be integrated onto the system. These solutions often define a package's location within the filesystem hierarchy whereas solutions to the software installation problem often define a package's internal directory structure. Often, both of these solutions include tools to help systems administrators install, distribute and remove packages according to a set of policies.

Policies may account for the efficient storage and distribution of software packages for a highly distributed heterogeneous network of computers or for a single computer.

Many take advantage of transparent remote network filesystem access such as NFS [26] and AFS [13]. Some use tools like *rdist*(1) [8] or the OpenDist package to duplicate software within a network.

Our experience with the Modules package indicates integrating software in discrete packages works best. Fortunately, it is becoming common practice that integrating software into a distributed network is done as separate packages. The clean separation of software packages and versions help to manage an ever-growing and fast-changing ocean of software.

Installation of software as separate packages has several advantages. Installation and removal of software becomes cleaner and errors or name conflicts become less probable. Multiple versions can be installed in parallel and can be selected individually. Software discovery becomes easier if a standard directory is used to install all packages. However software activation becomes more difficult because a user has to change the environment prior to starting an application.

Providing merged access directories (e.g., bin, lib, man) by linking all files from separate packages is one common approach to solve this problem. Users include these merged access directories in their path and have access to all installed software. However, if this merged directory is configured centrally, the user loses control over name conflict resolution or version selection. If he or she wants to use a different version, he or she still has to know where this software is installed and change his or her environment accordingly.

Also, a merged directory for commands quickly becomes huge. This increases the start-up time for shells with hash tables (like C shells) and the search time for commands in shells without hash tables. Symbolic links carry a performance penalty as well and require tedious maintenance.

Systems providing merged access directories to separately installed software packages include the "Corporate Software Bank" [24], LUDE [23], Depot (CMU) [15], its extension "Local Disk Depot" [17], and Xhier [22].

Xhier uses a user-owned merged access directory such that users can select which software packages are

linked. Users can select packages and versions.

The Depot (NIST) [21], Depot-Lite [16], `opt_depot` [18] install software as separate packages without specifying merged access directories. They rely on the user or other tools to provide for software activation. In fact, Depot-Lite uses the Modules package.

Also see John P. Rouillard and Richard B. Martin, "Depot-Lite: A Mechanism for Managing Software" for a discussion of several software integration solutions (such as a comparison of Depot-Lite with NIST Depot, CMU, CMU Ext, Ericsson, and Xhier).

The System V software management utilities (`pkgadd(1)`, `pkginfo(1)`, `pkgmk(1)`, ...) [12] and the emerging POSIX standard for software administration [25] address packaging and installation but not distributed integration of software.

No policies of how and where the software is to be integrated into the system are defined. Though, they recommend it should be possible to install software at any point in the filesystem hierarchy.

The main focus of LUDE is software integration. But, it also touches on installation and packaging of software. Software packages can be retrieved from LUDE ftp servers or tapes, and can be integrated into your installation by LUDE commands.

Activation

Software activation should hide integration details from the user. Users should be able to find out what software is available, what the software does and should have the ability to enable the software without knowing the details of its integration on the system. Of course, the user's favorite shell should be supported.

There are several tools purporting to solve the software activation problem. However, most of them solve only part of the problem. None of them provides the user with as flexible and as rich a feature set as the Modules package does. Most of the tools do not support software configuration management features like querying for available packages, dependency declarations, etc. See Table 1 for a comparison of features.

All of the solutions support initial setup of the user's environment during login by setting or altering environment variables. BNR [6], login-shared [5], and the Modules package use exactly one file per application to store the description of environment changes. login-shared stores the files with the applications, while BNR uses a configuration file to list their locations. The Modules package supports single or multiple arbitrarily nested directory structures to store configuration files.

Here we look at each solution independently in more detail.

Features	Modules Package	BNR	Ini	Login-Shared	Soft
Supported shells	sh, csh perl, emacs	sh, csh	sh, csh	csh	sh, csh
Freely Available	YES	NO	YES	YES	YES
On-the-fly Activate/Deactivate	YES	NO	YES	NO	NO
On-the-fly Reload/Refresh	YES	NO	YES	NO	YES
On-the-fly Exchange/Swap	YES	NO	NO	NO	NO
Define Alias/Shell Functions	YES	NO	NO	YES	NO
Set Local Shell Variables	NO	YES	NO	YES	NO
X Resource Manipulation	YES	NO	NO	NO	NO
Package Dependency Control	YES	NO	NO	NO	NO
Standard Configuration Language	YES	NO	NO	NO	NO
Hierarchical Package Naming	YES	NO	NO	NO	NO
Query Available and Active Packages	YES	NO	YES	NO	NO
Query Information About Packages	YES	NO	NO	NO	NO
Inherent Versioning Support	YES	NO	LIMITED	NO	NO
Symbolic Version Management	YES	NO	NO	NO	NO
Usage Tracing	YES	NO	NO	NO	NO

Table 1: Features of Software Activation Solutions

ini

ini [20] is for loading, unloading packages on-the-fly, as well as for querying available and loaded packages. Bourne and C shells are supported. *ini* uses a single directory to store configuration files describing environment changes needed to activate an application. Configuration information can be stored in a single file or on a per package basis. There is no hierarchical grouping of configuration files. The configuration files do not use a standard language. You can select to make packages available based on operating system, machine architecture or hostname. *ini* is implemented in *perl*(1) [11]. It is available free of charge for non-commercial use.

envv

envv [7] manipulates environment and local variables in a shell-independent manner. It supports Bourne and C shells. You can set and alter environment variables. When adding a component to a path list, you can define the position of the new component. You can also move components within existing path lists. *envv* has been used to write start-up scripts for applications that can be sourced by both Bourne and C shells. Once an application is activated by sourcing its start-up script, you cannot deactivate it unless you write a separate deactivation script that reverses the effects of the start-up script. *envv* is available free of charge.

Soft

Soft [3] uses a configuration file listing the packages a user plans to activate. For each entry in the user's configuration file, a database lists the necessary directories and variables to activate the application. Macros can band applications or other macros together. Shell script caches for both Bourne and C shells are created. These are sourced during login to construct the user's environment. Users modify their environment by editing the configuration file, executing an update tool, and finally sourcing the caches. There is no on-the-fly activation or deactivation of individual packages. *Soft* does not support querying for available packages. It uses its own language for its selection file. It is available free of charge.

login-shared

login-shared provides mechanisms for rapidly initializing an user's environment during login. C shell scripts for activating applications are kept with each software package (there is not a central repository). Applications can be activated interactively. A cache is created asynchronously for applications activated in the user's shell start-up file. The cache ensures during login that each environment variable is only set once. There are no commands to deactivate a package on-the-fly. A directory containing links for every command from every available package helps users locate packages. *login-shared* encourages users to activate many packages at login time. Because of this effect, a variant of the C shell was created for supporting a 4k-

character search path. With the Modules package's convenient on-the-fly activation and deactivation, the user's environment can remain small and thus requires no changes to system programs. *login-shared* only supports the C shells. It is available free of charge.

BNR Standard Login

BNR Standard Login supports the setup of the user's initial environment during login for Bourne and C shells. It uses configuration files (tables) to define how the environment is modified when activating an application. Environment and local variables can be set, unset or altered. Programs can be executed and the user can be queried to correctly set the TERM variable. Users can exclude applications from initialization. Also the *BNR Standard Login* supports delayed application configuration in order to make initial login faster. It does not support on-the-fly package activation and deactivation or software package discovery. It is not publicly available.

Our Modules Implementation

Initializing the Modules package and the *module*(1) command is accomplished by sourcing a shell-specific script into the shell. The script either creates an alias or a function in the user's shell which becomes the *module*(1) command. When invoked, the alias or function instructs the shell to evaluate the output of a program. The program is called *modulecmd*(1) and converts the user requests into shell directives.

Modulefiles are written shell-independently in Tcl and are interpreted by the *modulecmd*(1) program. Modulefiles can be loaded, unloaded or switched on-the-fly. Each modulefile describes the necessary changes to a user's environment in order to activate an application.

Typically, a modulefile is a small amount of code that sets or alters a few key shell environment variables such as PATH, MANPATH, etc. Using a rich programming language like Tcl allows for arbitrarily complex modulefiles that resolve issues like complex application dependencies, resource acquisition and application access and use policies.

From the user's perspective, changing the environment for one type of shell is exactly the same as changing the environment for another type of shell. One set of information takes care of every shell type.

Often, a pool of system modulefiles are shared by many users. The Modules package enables users to maintain their own collection of modulefiles that supplement or replace shared modulefiles. A search path for modulefiles controls locating modulefiles and can be dynamically manipulated.

It has become common practice to have one modulefile per revision of a software package. These modulefiles are stored in a directory named after the software package name. Referencing the directory structure selects a default version, yet a specific

version may be specified as well. These directories can be kept in single or multiple modulefile repositories. Arbitrarily nested directory structures can be utilized for storing modulefiles.

The Modules package source code is freely available. It is written in ANSI C and can be built on any modern UNIX machine with an ANSI C compiler (like the GNU C compiler).

Standard Features

Our Modules package supports the following features:

- Set or alter environment variables (pre-pending or appending path components)
- Environment variable optimization
- Alias definition (mapped to shell aliases or functions where appropriate)
- X-Resource manipulation
- A variety of shells and languages, e.g., Bourne shell, C shell, perl, and emacs
- Activate (load) / deactivate (unload) one or several packages in a single operation
- Unload all currently loaded packages (purge)
- Exchange (swap) packages or versions
- Refresh (reload) all currently loaded packages
- Display effects of loading a package on the environment
- Query available or loaded packages
- Per-package help and information
- Centralized and distributed repositories
- Alter modulefile repository search path
- Hierarchical application categorization
- Dependency declaration (conflicts and pre-requisites)
- Tcl as activation specification language
- Modules-specific Tcl functions to manipulate environment variables, aliases, and X-Resources in a shell independent manner to query the current operation mode (load, unload, info) and to query system information (machine name, operating system release, ...)

The Modules package can be used nearly everywhere for application activation. It is not limited to interactive use. Use it in your scripts (*sh*(1), *csh*(1), *perl*(1), or *emacs*(1)), in X11 startup scripts, X11 menu files, etc.

As an aside, we have decided not to implement environment caches for setup of the user's environment during login. The Modules package optimizes environment setting when activating multiple software packages with a single *module*(1) command.

Key New Features

Last year we began a major revision of the Modules package to add features that help the largest installations manage software. This revision is version 3.0 of the Modules package and is backward compatible with earlier releases.

Active Dependency Resolution

Previously, the loading or unloading of a modulefile is aborted if a conflict or a dependency is not met.

With Active Dependency Resolution, modulefiles can force modulefiles to be loaded and unloaded such that all conflicts and dependencies are resolved. This enables the activation of complete software systems containing several packages that are considered a set. Thus guaranteeing all software packages are loaded with tested and compatible versions by automatically unloading any conflicting packages and loading any missing packages.

Circular and conflicting dependencies are detected. In these cases, the user is informed of the error and the modulefiles are not loaded. Modulefile writers are responsible for ensuring these cases don't occur when specifying dependencies. Much like header file creation in C or C++, it is possible to create modulefiles with circular and invalid dependencies.

Symbolic Versions

A symbolic version can be assigned to a specific revision of a software package. The symbolic version turns a meaningless revision like 4.2.17.8 into a descriptive name like "beta", "current" or "old". For systems with multiple software packages assigning a symbolic version to the current revision reduces the maintenance effort for configuring software systems with many version of each package.

The user can easily spot which revision is released, which revision is beta, which revisions are old and which revision belongs to a specific system. The current assignment of a symbolic version to each software revision can be listed for future reference in case the user wants to exactly reproduce an environment after the symbolic names change.

Users have control of the revision selection as well as the assignment of symbolic versions to software revisions.

Autoload

Tcl provides for the autoloading of functions. If a function is called but is not defined in the current Tcl script, a list of directories is searched for Tcl libraries containing the function.

Autoloading eases the creation of libraries containing common functions. Modulefiles shrink since code is not duplicated leading to an overall reduction in maintenance costs. Displaying standard information for packages and platform detection are examples of common functions.

Apropos and Whatis

Similar to the UNIX *apropos*(1) and *whatis*(1) commands for manual pages, a one line description is printed for every available modulefile matching the given argument. With the number of packages installed at some sites reaching staggering heights, it is

no longer useful to retrieve information for every package one at a time. The new commands make it convenient to browse all of the installed software at large installations.

Logging and Usage Tracing

A network-wide logging of *module(1)* invocations using *syslog(3)* enables systems administrators to collect statistics on the usage of individual software packages. Knowing when a software package is no longer in use helps make the transition from old to new version of a software package smooth. It also helps track users of a software package for notification or interdepartmental charges.

Miscellaneous

- Oft-used package names or combinations can be abbreviated by defining a modulefile alias. For example, you can map OpenWindows/3.0 to ow3 to save keystrokes.
- Error messages are generated based on user-selectable levels. This helps novice users receive corrective information and keeps experts from complaining about verbosity.
- Configuration variables control enabling many of the new features without recompiling source code. For example, error and information messages can be configured to be printed to stdout, stderr, appended to files or passed to *syslog(3)*.
- A new test suite for the Modules package provides regression testing of most features. This helps maintain quality levels through the development of new features.

Experiences

There are some not so obvious issues that you might experience when introducing Modules. The ones we present here are culled from our own experiences and those expressed by other systems administrators using the Modules package.

“Continuum Breakdown”

The Modules package does not guarantee that the user's environment is restored to a previous state when deactivating modulefiles. Modulefiles may be very complex scripts that make it difficult or infeasible to exactly reverse their effect without storing the environment prior to their activation or storing the exact environment changes at each step. Altering the sequence of loading and unloading modulefiles adds yet more complexity.

In practice, the absence of an environment continuum has caused few or no problems for users.

“It's a Small World”

Our studies have found that most modulefiles are very small. Out of 250 different modulefiles, 75% contain less than 20 lines and only eight have between 100 and 150 command lines.

“You Win Some, You Lose Some”

A good software activation solution makes it convenient to install more software while software installation and maintenance is easier. In most cases, a user's PATH environment variable becomes shorter too. However, some users tend to activate every software package they might ever use in their shell's startup file. The environment is a limited resource. Especially the length of the PATH environment variable in some shells (notably the C shell). This limits the number of packages that can be active at the same time.

To avoid the limitations of the PATH environment variable, aliases that activate a software package using the *module(1)* command before calling the application can be defined. The same method can be applied for package bundles that load several packages with a single command. This is a form of autoloading that can reduce the number of packages that users need to load. It can also provide many of the features afforded by maintaining single directory with every command linked to its location without many of the pitfalls.

“Coins Have Two Sides”

Being able to efficiently manage large installations of numerous software packages in separate locations makes it convenient to provide several versions of the same software package. This does help users migrate to newer versions. They can switch back to an old version with a single *module(1)* command if the new version does not satisfy their needs (i.e., has new bugs). Although, this capability tends to make version transitions take more time.

Given this, there is less pressure for user's to migrate to new versions of the software by a given date. Some users stick with old versions (knowing how to work around bugs) so increased administrative pressure may need to be applied in order to get them to switch. However, we believe the flexibility gained far outweighs this side-effect.

“Start a New Shell, Do Not Pass...”

Though environment variables are inherited when starting a new shell, local variables and aliases are not. To work around this limitation, we added the update sub-command which resets the environment and reloads all currently active modulefiles.

An alternative solution we are pursuing is to have the Modules package reload those modulefiles with alias definitions when being initialized in a new shell.

Performance Considerations

There is a negligible performance penalty incurred at login when using the Modules package. On a SPARCstation 10/51 running Solaris 2.4 and version 3.0 of the Modules package, the average elapsed time to activate seven popular software packages (X11/R6, gnu, pbmplus, mtools, TeX, ppg, and Adobe's

Acrobat) is 0.6 seconds. Activating 21 software packages takes an average of 1.5 seconds. All of the software packages are activated with a single *module(1)* load command. We use the timing facilities of *tcsh(1)*. The times include any overhead from the usage trace feature.

Some users have experienced a new performance gain because not all applications must be activated during login. The dynamic activation features of the Modules package make it easy to activate less frequently used packages as they're needed.

We observe the performance penalties incurred by poorly configured or inattentive machines and networks are much higher than the overhead caused by using the Modules package.

"Think about the future..."

The Modules package based on Tcl is just one example of how the Modules concept can be applied to software management and activation. In this section, we venture into ideas which border on plans for future implementations of the Modules concept. We finish this section with a discussion of how Modules might be applied to the problem of configuring dynamically loadable Java [14] applications.

A Distributed Modules Server

The current implementation of Modules requires a separate filesystem distribution mechanism that permits all users of the Modules package to gain access to a common database of modulefiles. As the number of machines and the distance between the machines grows, keeping the database up-to-date becomes more difficult. We have begun preliminary work on the design and the development of a distributed Modules server. Such a server would assist with the problems of updating and distributing modulefiles to a wide range of machines and domains.

Software Discovery

Beyond the software discovery mechanisms available in the Modules package, we see a Modules server permitting collaboration between different administrative domains within a company. Individual domains run their own server which communicates with other servers within a company. Queries for available software within one domain can be modified to extend to other cooperating domains. Through NFS or some other file access method, an application in one domain may be discovered and made available to the user. Note that software discovery and activation in this manner is much like a selecting an HTML link on a Web page.

Software Information

We expect the Modules server to act as a software information and documentation tool like *csdsdb* [19]. Users should be able to query for information about each software package like:

- Maintainer's contact address

- Licensing information
- Potential source code availability
- Revision history
- Record of applied patches
- Planned upgrade information

Activating or retrieving information about the software can use the same user interface.

Programmers Interface

We have already seen the development of a wide array of tools that use and augment the Modules package. With the expanded capabilities of a Modules server, we see such tools becoming more prevalent. Providing an application programmer interface to the query capabilities, environment manipulation mechanisms and an ability to programmatically setup a new program's environment via the Modules server is an important goal.

For example, building Modules interaction into a UNIX shell reduces the need to form a traditional filesystem-based search path. When the shell is initialized, the Modules server can download application names into the shell's hash table directly without having to mount and search each filesystem. Upon invocation of any of these names (or aliases), a wide variety of environment preparation can take place. Furthermore, the shell can dynamically receive new program names and information from the Modules server as they become available or change.

User Profiles and Release Management

Based upon a user's organization or a user's profile, the list of available software and information can be customized. For example, certain applications (oh, say *salarytool(1)*) may be available to managers in an organization but not to every employee.

Profiles can also be used to remove clutter. Some users may not be interested in knowing about the wide assortment of CAD software available on the network. Being able to configure this software out of the viewable list of software assists novice users who may be overwhelmed by the large amount of software available on the network at a typical large company.

Often, applications are not available on all the hardware platforms within a company. The server can flag software packages that are unavailable on the client's hardware platform or can make them appear unavailable to the client.

Hiding software packages that are not-yet-released is another variation. A list of users, e.g., beta-testers, will see these software packages listed while other users will not be tempted by potentially unstable software.

Performance and Complexity

We expect the Modules server to increase the performance of processing modulefiles because commonly used modulefiles can be cached and potentially pre-compiled for commonly used shells. Redundant servers can ensure availability of the Modules service.

Dependencies between modulefiles can be stored in the servers' database to discover indirect dependencies or conflicts. Dependency and conflict matrices can be provided as well, e.g., to assist in discovering conflicting dependencies.

Using DNS as Modules Server

Another path we are considering is using the existing Internet Domain Name System (DNS) protocol and query mechanisms to provide some of the aforementioned Modules server capabilities. DNS permits storing arbitrary amounts of text and other information for a given domain name. We are investigating the potential of using a reserved domain name within an intranet to provide the existing *module(1)* capabilities. Special domain names can specify query requests for available modulefiles. Once the list has been received, the client can query the DNS server for more information about specific packages.

Due to the static nature of DNS records, a Modules server may still be necessary for some of the more advanced query capabilities. Although, it may be possible to use DNS to help locate the appropriate Modules server for a given application or domain.

"The Taming of Java"

Much of Modules is about preparing the system's environment prior to the execution of an application. Currently, modulefiles contain the steps necessary to activate an application. This includes problems like resource allocation and verification, environment modification and dependency and conflict resolution. As more complex applications are developed for downloading over the Internet or over intranets, the traditional problems of integrating software onto the system arise.

Applications should not be responsible for handling configuration issues on the local host. As it stands today, Java applets request additional resources and other applets which may or may not be available to the local machine. The execution of the applet fails if these capabilities are not obtained – potentially after much of the applet has been downloaded. For example, certain types of or portions of applications will not work through a firewall. A user may begin loading such an application over the network and even begin executing it before discovering the missing dependency. If activating a particular feature of the application invokes an unsupported access to outside the firewall, then it depends on how well the application handles the failure and the nature of the work as to whether the failure will have a minimal or catastrophic affect on the user's work.

We envision a small modulefile-like Java program being downloaded into the local host prior to downloading and executing an application. Using a set of well-defined system configuration interfaces and tests as well as all the features of Java, the modulefile can verify the capabilities of the local host. Also, the modulefile can verify resolution of dependencies on

other applets for the application. Once the modulefile has verified the host is configured such that it is likely to successfully execute the application does the transfer of the application begin.

Some sites may have different network restrictions and configurations as well as different policies with regard to downloading Java applications. The modulefile would be able to scope out these configuration issues prior to downloading the application in addition to potentially modifying the behavior of the application to suit local configuration parameters. With network bandwidth at a premium and Java applications tackling larger problems, using Modules in this fashion reduces the likelihood of wasting network bandwidth on attempting to load applications that won't execute.

As Java applications become more complex they will have the same tendency to depend on particular versions of other applets or Java libraries. The modulefile would be able to verify the local host doesn't have incompatible versions loaded. For a while, other dependencies may include the installation of a particular set of native methods on the local host.

The modulefile doesn't have to be static. It can interface with the user to suggest modifications to the local host prior to loading the application.

Like the modulefiles used in the Modules package, the pre-loaded modulefiles can provide information about the application that helps the user determine if he or she wants to continue the download. Furthermore, these pre-loaded modulefiles can handle issues like licensing and monetary exchange prior to downloading the application. In such a situation, the user might enjoy an indication of whether the application will execute on his or her machine.

We have just begun to scratch the surface of an implementation based on the ideas presented in this section. Some number of security and certification issues will need to be resolved as well as a more concrete understanding of the necessary configuration parameters a local host can make available to the modulefile before any implementation is complete.

Summary

In abstracting the user from the details of software installation, distribution and integration, Modules is a compelling solution to the software activation and management problem for large-scale distributed networks. We have contrasted the Modules package with other solutions to the software activation problem. In addition, we have introduced the reader to the features of the latest release of the Modules package as well as ideas for utilizing future capabilities of a Modules server.

Lastly, we have presented how the Modules concept can be exploited to manage the burgeoning configuration problems surrounding a swelling sea of

increasingly complex Java applications.

Availability

Visit Modules' home page at <http://www.modules.org/> for latest information on Modules.

Modules is freely available from <ftp://ftp.modules.org/pub/Modules/> or from your friendly Tcl archive in your neighborhood, like <ftp://ftp.neosoft.com/tcl/>.

You can subscribe to the Modules interest mailing list, by sending the line "subscribe modules-interest <your-email-address>" in the message body to majordomo@modules.org.

Acknowledgments

We would like to thank a number of individuals who have made significant contributions in their support of the Modules package: Tony Bennett, Maureen Chew, Richard Elling, Leif Hedstrom, Doug Kubel, Don Libes, Ken Manheimer, Marty McLean, Phillip Moore and John Rouillard.

Development of the 3.0 version of the Modules package has been funded in part by the Semiconductor Group of Siemens AG. Much of the new features were implemented by Jens Hamisch, Strawberry EDV-Systeme GmbH, Munich. We are especially thankful for his contributions during the specification of the new features and his solid implementation. We also thank Connect! GmbH, Munich for providing ftp and web space.

John thanks all of his co-workers through the years for supporting the use of the Modules package and putting up with the bugs and problems during its development. John thanks his parents for their continuing support of all his travels and endeavors.

Peter thanks SAM for their *Never Ending Story*. Also, he would like to thank Johnny Clegg & Savuka for their album "Heat, Dust & Dreams" which helped him to keep working on Modules and this paper.

Author Information

John L. Furlani is currently technical lead of the Solaris XIL Imaging and Video development team at SunSoft, Inc. in Mt. View, CA. John received a Bachelor of Science in Electrical and Computer Engineering from the University of South Carolina at Columbia in 1990. While employed by Sun in North Carolina, he received a Masters of Science in Computer Science from Duke University in 1994. He was a systems administrator at USC and the Naval Research Laboratory in Washington, D.C. during his undergraduate college years. Upon graduation, John joined Sun Microsystems, Inc. as the systems administrator for Sun's North Carolina Development Center in Research Triangle Park, North Carolina. John enjoys playing the keyboard and the bassoon, hiking, traveling, snow skiing, cycling, rollerblading, cooking,

eating great food and wine tasting. John can be reached via e-mail at j.furlani@ieee.org.

Peter W. Osel received his diploma in electrical engineering from the Technische Universität München (TUM) in 1985. For three years he worked at corporate research of Siemens AG, where he developed tools for ECAD of Integrated Circuits. From 1988 until June 1996 he was working for the Semiconductor Division of Siemens. He was responsible for worldwide integration and distribution of the CAD system, as well as the development of central tools, and the coordination of the development sites' system environments. Since July 1996 he is working for Siemens Business Services GmbH & Co OHG in the department that administrates all workstations of the Semiconductor Division of Siemens. In September 1996 he moved from Munich, Germany, to Cupertino, CA, where he designs, implements and administrates the compute environment for the development and marketing of Siemens' new microcontroller family. Reach Peter at Siemens Business Services GmbH & Co OHG, SBS DS 33, Postfach 801709, D-81617 München, Germany; or at Siemens Components, Inc., 10950 North Tantau Avenue, Cupertino, CA 95014; or by e-mail at pwo@HL.Siemens.DE, or see his Web page at <http://www.ConnectDE.NET/~pwo/>.

References

- [1] John L. Furlani, "Modules: Providing a Flexible User Environment", *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*, pp. 141-152, San Diego, CA, September 30 - October 3, 1991.
- [2] John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison Wesley Publishing Company, Inc., ISBN 0-201-63337-X, 1994.
- [3] Rémy Evard and Robert Leslie, "Soft: A Software Environment Abstraction Mechanism", *Proceedings of the Eighth Systems Administration Conference (LISA VIII)*, pp. 65-74, San Diego, CA, September 19-23, 1994.
- [4] Richard Elling, Matthew Long, "user-setup: A system for Custom Configuration of User Environments, or Helping Users Help Themselves", *Proceedings of the Sixth Systems Administration Conference (LISA VI)*, pp. 215-223, Long Beach, CA, October 19-23, 1992.
- [5] Carl Hauser, "Speeding Up UNIX Login by Caching the Initial Environment", *Proceedings of the Eighth Systems Administration Conference (LISA VIII)*, pp. 117-124, San Diego, CA, September 19-23, 1994.
- [6] Christopher Rath, "The BNR Standard Login (A Login Configuration Manager)", *Proceedings of the Eighth Systems Administration Conference (LISA VIII)*, pp. 125-138, San Diego, CA, September 19-23, 1994.
- [7] David F. Skoll, "envv - manipulate environment

- variables in a shell-independent manner", UNIX man page and source code for version 1.6 of the application, July 1995.
- [8] Michael A. Cooper, "Overhauling Rdist for the '90s", *Proceedings of the Sixth Systems Administration Conference (LISA VI)*, pp. 175-188, Long Beach, CA, October 19-23, 1992.
 - [9] Peter W. Osel, Wilfried Günsheimer, "OpenDist - Incremental Software Distribution", *Proceedings of the Ninth Systems Administration Conference (LISA IX)*, pp. 181-193, Monterey, CA, September 17-22, 1995.
 - [10] Cray Research, Inc. Products. <http://www.cray.com/PUBLIC/product-info/sw/>.
 - [11] Larry Wall and Randal L. Schwartz, *Programming perl*, O'Reilly & Associates, Inc., Sebastopol, CA, 1991.
 - [12] System V software management utilities, Solaris 2.5 manual pages *pkgadd(1)*, *pkgmk(1)*, *pkg-trans(1)* etc.
 - [13] *AFS distributed filesystem FAQ*. <http://www.cis.ohio-state.edu/hypertext/faq/usenet/afs-faq/faq.html>.
 - [14] James Gosling, Henry McGilton, "The Java Language Environment", A White Paper from Sun Microsystems, Inc., October 1995. http://java.sun.com/doc/language_environment/.
 - [15] Wallace Colyer and Walter Wong, "Depot: A Tool for Managing Software Environments", *Proceedings of the Sixth Systems Administration Conference (LISA VI)*, pp. 151-162, Long Beach, CA, October 19-23, 1992.
 - [16] John P. Rouillard and Richard B. Martin, "Depot-Lite: A Mechanism for Managing Software", *Proceedings of the Eighth Systems Administration Conference (LISA VIII)*, pp. 83-91, San Diego, CA, September 19-23, 1994.
 - [17] Walter C. Wong, "Local Disk Depot - Customizing the Software Environment", *Proceedings of the Seventh Systems Administration Conference (LISA VII)*, pp. 51-55, Monterey, CA, November 1-5, 1993.
 - [18] opt_depot - http://www.arlut.utexas.edu/opt_depot/opt_depot.html.
 - [19] *csdsdb - Computer Science Division Software Database*, <http://www.arlut.utexas.edu/csd/csdsdb/>.
 - [20] ini - GeNUA GmbH, <ftp://www.genua.de/tools/ini.tar.gz>.
 - [21] Kenneth Manheimer, Barry A. Warsaw, Stephen N. Clark, Walter Rowe, "The Depot: A Framework for Sharing Software Installation Across Organizational and UNIX Platform boundaries", *Proceedings of the Fourth Large Installation Systems Administrator's Conference*, pp. 37-46, Colorado Springs, CO, October 18-19, 1990.
 - [22] John Sellens, "Software Maintenance in a Campus Environment: The Xhier Approach", *Proceedings of the Fifth Large Installation Systems Administration Conference (LISA V)*, pp. 21-28, San Diego, CA, September 30 - October 3, 1991.
 - [23] Michel Dagenais, Stéphane Boucher, Robert Gérin-Lajoie, Pierre Laplante, Pierre Mailhot, "LUDE: A Distributed Software Library", *Proceedings of the Seventh Systems Administration Conference (LISA VII)*, pp. 25-32, Monterey, CA, November 1-5, 1993.
 - [24] Steven W. Lodin, "The Corporate Software Bank", *Proceedings of the Seventh Systems Administration Conference (LISA VII)*, pp. 33-42, Monterey, CA, November 1-5, 1993.
 - [25] Barrie Archer, "Towards a POSIX Standard for Software Administration", *Proceedings of the Seventh Systems Administration Conference (LISA VII)*, pp. 67-79, Monterey, CA, November 1-5, 1993.
 - [26] Sandberg, R., D. Goldberg, S. Kleiman, D. Walsh, B. Lyon, "Design and Implementation of the Sun Network Filesystem," *USENIX Conference Proceedings*, USENIX Association, Berkeley, CA, Summer 1985.

SLINK: Simple, Effective Filesystem Maintenance Abstractions for Community-Based Administration

Alva L. Couch – Tufts University

ABSTRACT

We manage several large UNIX program repositories through a community effort of volunteerism and advocacy. Our effort requires a carefully crafted interplay between administrative policy and tools that operate within the limits of that policy. Rather than restricting administrators' actions, our tools reinforce their own use by making it easier and more effective to comply with policy than to dissent. Our tool SLINK provides a small number of commands that aid in synthesizing user environments from sets of disjoint software package trees. SLINK's commands, while more powerful than typical UNIX commands, refuse to violate predefined policy restrictions, thus protecting the user environment even from mistakes of root users. Our administrative policy and SLINK allow us to employ an arbitrarily large number of volunteer administrators without degrading system behavior or utilizing large amounts of staff time.

Introduction

We maintain a medium-scale, heterogeneous, departmental, academic UNIX network (50 stations, 1000 users) for the Department of Electrical Engineering and Computer Science at Tufts University. Our academic mission requires us to provide the most current versions of multitudes of software packages for Sun, DEC, and SGI systems. Our program repositories contain hundreds of programs, fill 10 gigabytes, are constantly being revised and updated, and encompass almost every form of software from freeware for simple document processing to complex commercial systems for simulation and computer-aided design.

We accomplish this mission with almost no staff involvement, due to a novel repository management approach based upon community involvement, user empowerment, administrator volunteerism, and software advocacy. Our shortage of administrative staffing is offset by availability of untrained but trustworthy, competent, and enthusiastic student and faculty help. Conscious of the many problems in scaling a standard managerial approach to student help [7], we developed instead an approach in which maintainers form a loosely coupled community bound together by policy much resembling a community charter. This policy is supported by tools that make it easier to comply with policy than to dissent. Thus we maintain system integrity and consistency not by force, but by making compliance the path of least resistance.

Previous approaches we tried for repository management caused tremendous frustration for us. Simply utilizing */usr/local* as intended led to constant problems requiring administrative intervention. Student workers often installed conflicting versions of the same package so that neither version worked. We

never quite knew who installed what or when. Due to rapid turnover of software and the need to constantly update to new revisions, we were always breaking one software package or another, with no ability to control changes or to test software before installation. Worst, */usr/local* frequently filled up with files of unclear, undocumented function that we could not delete without unacceptable risk of breaking other, unknown programs.

Six years ago, we began working on solving this problem. We started by trying to install all programs in separate subtrees of identical structure, and wrote a very simple tool to build an image of the union of all subtrees that users could employ. This tool was a complete failure. Very little of the software we use would adapt to being installed in a 'standard' way, and the tool remained mostly unused. We spent more time adapting software to this standard form than we would have spent cleaning up a typical */usr/local*. We learned from bitter experience that any approach we use has to be very flexible and tolerant of deviance in software packages.

This flexibility required that we adopt a different philosophy about the role of tools in our maintenance strategy than is typical in modern practice. Most tools for repository management *enforce* standards by refusing to function unless requirements are met. Our administrators could not function in such an enforced environment, because the software we were trying to maintain was too difficult to adapt. Because of our site's volatility, budget, and need to keep up-to-date, we do not have the time to fight with software packages. Any strategy we adopt must allow us to break the rules instead of wasting time trying to comply.

If we allow breaking the rules, there must then be a way to maintain order in the ensuing chaos. We accomplish this by creating tools that, instead of limiting the administrator, make it easier to install software in the preferred way than in other ways. These tools do not enforce, but *reinforce* our policies. A tool that reinforces policy works whether or not that policy is followed, but 'works better' when policy is followed than when it is ignored. This means that everyone, whether compliant or not, will be able to perform useful work, but that compliant administrators will receive more positive reinforcement in the form of completed work.

Background

A *program repository* is a filesystem wherein programs to be executed on a network are stored. Repositories can be formatted so that programs within them are ready to run, and mounted on remote systems via the network file system (NFS) or some similar mechanism. Repositories can also consist of information to be copied, through some mechanism, to remote hosts before being executed. Our repositories are ready to run, NFS-mounted by nearby hosts, and can be copied to remote hosts, one package at a time.

Repository management refers to the activity of maintaining such a repository over time, which comprises adding and deleting programs and revisions of programs, and distributing software changes to remote systems where applicable. A management strategy should define appropriate *change control*, whereby the user's software environment is changed in an orderly, predictable, and reversible fashion as a result of software updates. The strategy should also avoid repository corruption or rot [5], in which the repository fills up with old files that have no function but cannot be efficiently located and deleted.

Many tools, including vendor-supplied software installation scripts, manage software repositories and avoid repository rot through a *package* structure [8]. Each software distribution occupies a separate filesystem tree called a package. Typically all such package trees exhibit parallel structure, so that each has a section for user commands, libraries, and documentation the user might need. Subject to obeying interdependencies between packages, package subtrees can be independently copied to remote hosts to provide custom software environments on each host. But simply copying them is not enough; to utilize the package the user must have access to those commands, libraries, and documentation.

The most common way these packages are made available to a user is through *environment synthesis*. The various packages are combined through some mechanism into a coherent user environment having single program, library, and manual directories that can then be included in each user's search paths. Usually this is accomplished through symbolic links [1, 3,

8, 9, 10, 11, 12], but it may be accomplished by copying files, as is done in several vendor-supplied software distribution systems.

Many developers have extended the basic distribution framework of Depot [8], whose basic capabilities include package definition, distribution, and environment synthesis using symbolic links. Although initially Depot expected all packages to have a predefined structure and obey particular naming conventions, now the structure of each package, including a list of files the user should see, may be declared within a file in each package. In Depot and its relatives [1, 3, 10, 12], changes are controlled and the user environment is protected from errors by a transaction commitment process, in which the proposed new environment is checked for lack of conflicts before the old environment is erased and a new one created in its place. A conflict occurs when one file in the environment has two possible definitions, usually due to an installer error. This strategy requires that the environment, as a filesystem subtree, be completely under the control of the software tool that destroys and recreates it.

Repository distribution tools like Cicero [2] (and some more recent variants of the Depot approach) take transaction commitment one step further, by giving the administrator control over the sequence of individual software installation transactions and the ability to undo transactions to recover from installation errors. This means that Cicero is not limited to installing packages in designated filesystem subtrees, but can safely undertake changes of a broader scale, such as changing files in */etc*.

SLINK

Our tool SLINK [5, 6] concentrates on simplifying the process of environment synthesis. SLINK is a freely available Perl5 script and library that is portable to a majority of UNIX systems. It has no distribution capabilities like Cicero or Depot relatives, and instead operates on a tree of packages that has already been distributed to the target host by some other distribution mechanism, such as NFS or rdist [4]. The version of SLINK described in this paper has been substantially modified from that described in [5], mainly by improving declaration file syntax and by adding the virtual protection mechanism for filesystems described below.

SLINK differs from other approaches in several important ways. SLINK attempts to make the environment synthesis task as simple as possible for an administrator to perform, so that administrators can be trained very quickly to modify environments. SLINK operates mainly by interpreting a single configuration file that tells it what should be true of the target system. One can create package-specific configuration files but this is not required. SLINK constructs environments *incrementally*, by making the changes necessary to install one package at a time, rather than

creating an environment from scratch every time as in most Depot-like strategies. SLINK does not need to maintain complete control over the environment, but can merge its changes into an existing environment as easily as it can build a new one.

SLINK has five basic commands that control the user environment by specifying parallel structure between the package trees in which administrators install software and the image trees that normal users utilize. Each command is a structural assertion about the similarity between two UNIX filesystem subtrees, that specifies either that the contents of one be contained or not be contained within another. Commands recursively update the structure of filesystem subtrees to arbitrary depth, so that whole file hierarchies can be duplicated with a single command. These commands are general-purpose and can be used for a variety of replication tasks other than repository maintenance.

The command

```
link /loc/lang/perl/bin /local/bin
```

says that */local/bin* contains the contents of */loc/lang/perl/bin*, plus perhaps more files, by utilizing symbolic links to point from */local/bin* to */loc/lang/perl/bin*. The command

```
unlink /loc/lang/perl/bin /local/bin
```

undoes the *link* command, specifically unlinking any links in */local/bin* that happen to point to corresponding files in */loc/lang/perl/bin*. The *link* and *unlink* commands implement environment synthesis, including both installation and deletion of software. They operate recursively on whole filesystem subtrees, so that one can install *all* of *perl*'s files in */local* by typing

```
link /loc/lang/perl /local
```

This is done by optimal use of symbolic links; for details, see [5]. This syntax has changed from that described in [5], due to difficulty in reading the original syntax.

The current version of SLINK also includes commands for normal copying of files from one place to another. The command

```
copy /loc/lang/perl/bin /local/bin
```

does the same thing as *link*, except that it copies files rather than linking them, while the command

```
uncopy /loc/lang/perl/bin /local/bin
```

removes files that are identical (in all respects, including owner, mode, timestamps, and contents) to those in */loc/lang/perl/bin*. This replaces *cp -r*, *tar*, and other more primitive ways of copying files. However, copying only works perfectly for root users, and of course does not preserve owner or group of each file when invoked by normal users.

The commands *link*, *unlink*, *copy*, and *uncopy* function more like structural assertions than

commands, and do nothing if the desired condition already exists. The commands *unlink* and *uncopy* only undo exactly matching *link* and *copy* commands; these will not delete files in the user's environment that happen to have the same name as files in the package tree but were replicated from another package or source. This means that *unlink* and *uncopy* can be repeated even if the affected files are replaced by files of the same name but from newer revisions; *unlink* and *uncopy* will not destroy these newer files.

The command

```
destroy /local/bin
```

will unconditionally destroy */local/bin* if allowed. This is mainly a cleanup command.

As SLINK does nothing if a series of command assertions are already true, it can safely be used to verify or change the configuration of systems while users are using them. SLINK may, in the course of an incremental change, turn a symbolic link into a directory of links, but will never do the reverse by default, so that no user can be deprived of a current directory through SLINK's actions.

Virtual protections

SLINK's commands are very powerful and dangerous in the hands of the uninitiated if left unrestricted. In particular, a root user could destroy the whole disk image by typing '*destroy /*', or corrupt the */usr* partition by typing '*link /usr*'. Such powerful commands must be tempered with limits so that SLINK is safe to use. This is done through a *virtual protection* scheme that augments the normal *physical* UNIX filesystem protection scheme. The virtual protection scheme is new to SLINK 5.0; in the previous version protection directives applied only to individual commands. We realized through bitter experience that this was an unsafe practice.

A virtual protection is a voluntary mechanism implemented by a software tool to protect against unwise actions that are perhaps physically possible. While physical protections are enforced as usual, by UNIX, virtual protections are enforced by SLINK itself. In the performance of its duties SLINK will refuse to modify any virtually protected subtree. Of course, the administrator can work around this by making changes manually, given enough physical privilege. But this is considerably more difficult than using SLINK commands.

Our virtual protection scheme is designed to be as simple as possible while supporting common administrative operations for repository management and filesystem maintenance. There are currently five levels of virtual protection we support:

1. freeze – prohibit all changes to this filesystem.
2. protect – allow only the addition of files, links, and directories.

3. *relink* – allow the removal and recreation of symbolic links.
4. *redirect* – allow the removal and recreation of links and directories consisting only of links.
5. *replace* – allow any change, including removing files or directories containing files.

These protections are the simplest scheme we could design within the context of SLINK's mission. The reasons for protections of *freeze*, *protect*, and *replace* are obvious to any administrator, allowing one to completely protect a filesystem, allow additions only, or allow all changes. The protections *relink* and *redirect* are specific to the needs of SLINK in incrementally maintaining repositories.

The *relink* protection allows one to protect file and directory structure while changing links. This allows one to build a filesystem consisting of files and directories, and then augment it with links that cannot overwrite those files and directories, so that the filesystem becomes a combination of an unchanging core of files and a relatively fluid set of links. We use this mechanism to protect very important parts of */local* that should never be overwritten, such as */local/bin/perl* and */local/bin/tcsh* (without which no users can work), by making them actual files.

SLINK updates images by promoting symbolic links pointing to directories into directories of links to the contents of those directories, where needed. Over time, *unlink* requests can result in directories of symbolic links that are indeed equivalent with single links. The *redirect* protection tells SLINK that it can change these directories of links back into single links on command. One does not always want SLINK to do this; certain directories must exist whether or not they are equivalent with single links or empty, e.g., lock directories.

Protections for all filesystems are specified in a single file for each machine, which is kept separate from SLINK's configuration file, though the configuration file can also contain protection directives mixed with regular SLINK commands. The file is read by a Perl5 library function that then communicates policy to SLINK.

A typical machine's protections might be:

```
freeze /
protect /usr
redirect /local
freeze /local/man/cat*
```

The protection of a path is the protection of the longest prefix of the path with an explicitly defined protection. In the above scheme, */local/man* has the protection '*redirect*' because its longest prefix with an explicit protection is */local*, while */etc* has the protection '*freeze*' (because its longest explicitly protected prefix is */*). The star convention works as in *sh* and is checked against pathnames dynamically as SLINK executes, so that paths SLINK creates are protected

once created. For example, if */local/man/cat5* does not exist, when it is created, it will be treated as frozen. These conventions were adopted to make virtual protection files as brief as possible.

Virtual protections keep SLINK's brute power from destroying the system, even for root users, by stopping dangerous acts. For example, the protections

```
freeze /
relink /local
freeze /local/man/cat*
```

protect systems from student mistakes by only allowing linking in */local* and its subtrees, except for the formatted manual page directories */local/man/cat1*, etc. This keeps SLINK from placing links into those directories that might conflict with the proper function of the *man* and *catman* commands. So, if naive administrators exclusively utilize SLINK, they can affect nothing but the user's environment, no matter what their privilege or what SLINK commands they issue.

A virtual protection failure is not an error, however, because very often one wishes to structure a filesystem by copying everything from another parallel one, except for specific things. For example, the commands:

```
freeze /
protect /usr
freeze /usr/spool
link /lusr /usr
```

add links from */usr* to any file absent from */usr* that is in */lusr*, ignoring */usr/spool* for obvious reasons. This allows us to 'fill out' a small */usr* partition with files from a remotely mounted full copy */lusr*, a hack we use quite frequently when pressed for space on small or old workstations.

As another example, the commands

```
freeze /
relink /local
destroy /local
```

will destroy every symbolic link in */local* while leaving files and directories alone, because the '*relink*' protection prohibits SLINK from removing them. Using this sequence, one can restart SLINK from scratch in building a repository, in the same manner as Depot.

SLINK's *copy* command must be used with discretion. Its results are not as easily documentable or reversible as those for the *link* command, and we discourage its use except for special purposes, like embedding changes within an image of a read-only filesystem. Suppose, for example, that we have a CD-ROM containing UNIX manual pages that we'd like to augment with our own manual pages. We can mount the CD-ROM as */cdrom*, put our manual page hierarchy into */myman*, make */usr/man* an empty directory, and then instruct SLINK to:

```
freeze /
replace /usr/man
link /cdrom /usr/man
copy /myman /usr/man
```

The copy operation will override the contents of */cdrom* with those of */myman*, creating a union of the two hierarchies in */usr/man*. Of course, we could do this with manual paths, but it serves as a demonstration of SLINK's power. One can, for example, create a modified copy of a read-only */usr* in the same way, whether mounted from a CD-ROM or via NFS.

SLINK limitations

There are also several things that SLINK does not do that other tools support. Because it is an incremental strategy, SLINK will function even in the presence of conflicts. After several conflicting commands, the source of a file with conflicts is the one specified most recently. SLINK will inform the user of any conflicts it finds, but one usually has to execute a set of configuration commands twice in order to determine whether there is a persistent conflict in instructions or just a transient conflict between older and newer software versions. SLINK is completely tolerant of errors in configuration and will happily ignore all configuration lines it cannot parse, implementing the ones it can act upon. In general, the tool operates in a much less constrained (or paranoid) manner than Depot, Cicero, and their relatives.

Because SLINK is an incremental environment synthesizer and does not have the luxury of creating a whole new environment each time it is invoked, it must also support cleanup functions not supported in other tools. There are functions that scan user environments for dangling links, files, and other exceptional conditions, and functions that clean up after installation mistakes. There are functions that optimize link structure to utilize a minimum of symlinks. No matter how easy SLINK is to use for naive administrators, a relatively skilled administrator is still needed to perform these complex functions.

SLINK is capable of synthesizing very complex user environments. The virtual protection system, while certainly useful, can lead to unpredictable results in the hands of the uninitiated. To aid in debugging, we provide a separate script *slinkls* whose purpose is to show the structure of what SLINK created in an easily understandable form. This program takes a large amount of computer time to look at an environment and express it in terms of the SLINK commands needed to create it. Currently *slinkls* only describes linked structures, because it does not have enough information to describe *copy* operations. SLINK's incremental strategy can backfire if used carelessly, and *slinkls* does not fully address the problems that can arise.

SLINK's assertions work on filesystem trees as easily as upon individual files, so that the number of

SLINK commands needed to install a package varies directly with the lack of parallel structure between package and image. If an installer forms a package tree that is exactly parallel in structure with the image tree, the whole package can be made available with a single SLINK command. Lack of parallelism means that several commands may be required. In extreme cases, especially when installing commercial software, every file may need to be linked with an individual command. Nonetheless, it is always possible to link any package into the image, though well-structured packages are always easier to link than badly structured ones. Disciplined administrators are 'rewarded' with a painless incorporation process, while undisciplined ones are 'punished' by having to write more SLINK commands.

Our experience with less flexible tools has led us to believe that repository administration is easier to perform and to teach when reinforced than when enforced. In practice, doing something 'wrong' has very little impact; novices' mistakes can be easily repaired by a few simple SLINK directives. This means, however, that every administrator must refer to a policy document as well as tool documentation. We find such documents to be shorter, easier to write, and easier to understand than docs for a complex tool.

Our Policy

Our policy requires administrators to install software in disjoint package trees, which are then combined to form an image filesystem that the user sees on a particular workstation, utilizing a combination of symbolic links and file copying. This image contains all commands, libraries, and documentation the user might want, with one directory per type of information, similar to the structure of */usr/local*, e.g.

Our policy for package installation is fairly simple:

1. To the extent possible, install software in disjoint subtrees, one per software distribution.
2. To the extent possible, mimic the structure of */usr/local* in those subtrees.
3. To the extent possible, programs and libraries in packages should refer to files in their own package by their full path names in the package tree.
4. To the extent possible, programs and libraries should refer to files in other packages as if they are installed in */usr/local* or an equivalent public space, except when a program or library depends upon a specific revision of a file, in which case that file should be referred to by its full pathname in the package hierarchy.
5. To the extent possible, keep files not needed during package operation separate from needed files so unneeded files can be deleted.

An ideal package tree contains a *bin* for commands, a *lib* for libraries, a *man* for manual pages, a

src for source code, etc. It may also contain an *etc*, an *sbin*, etc, as appropriate.

In this policy, all requirements are voluntary, and designed to be taught to aspiring administrators in one day of instruction, along with instructions on using SLINK. The policy above is the first level of several levels of detail, written to be easily understood by novices. Advanced administrators can take the time to delve deeper into its subtleties.

Delegation

Our policy and SLINK simply suggest the appropriate course without enforcing it. A particular administrator's powers and privilege are determined instead by normal UNIX filesystem protections. We maintain order in the community by carefully delegating both increasing privilege and trust relative to experience and prior performance.

There are five levels of delegation: beginning, novice, apprentice, root, and netadmin. Beginning administrators are asked to install packages in their home directories or in a practice tree until they are proficient enough to work in the real package tree. Novices are given ownership of particular package directories in the true package tree and asked to install particular packages in each. When they prove competent, they become apprentices and are given membership in a group with privileges to modify the package tree and user image tree. When apprentices prove competent, they are given root privileges and additional instruction on modifying files in places other than the package and image trees. When root users become proficient, they are trained in network administration and control of global network configuration.

SLINK is not a set-user-id program. It always operates with the privileges of the invoker. Without appropriate group privileges, SLINK will do nothing for novices. Apprentices can execute SLINK or modify the user environment directly in emergencies, but cannot modify system files outside the trees made available to normal users. Virtual protections are not limits, but reminders of one's responsibility to the community. For this reason, any administrator is allowed to override virtual protections in SLINK's configuration file (though the master file of virtual protections cannot be changed). Anyone who has the physical ability to make such changes might as well be trusted to do so through SLINK rather than manually.

Security

System integrity and security are serious problems in any system maintained by extensive delegation. Our strategy has always been to evaluate people carefully before increasing their privilege or responsibility. To avoid inadvertent system corruption, we typically only allow one undergraduate to hold root privileges at a time, so that conflicting actions between two students are less likely. While this makes root

privileges a much sought trophy, we continually stress that these privileges are a responsibility, not a power.

In six years and for about 20 students, we have guessed wrong only once. One student administrator did not take his responsibility seriously, and violated the privacy of other students. He was suspended from the University for his actions.

Of course, there have been many times when a new software package did not function, or when a student's actions disrupted service for a short time. Fortunately, SLINK allows one to easily reinstall older versions by changing the configuration file, so we have experienced minimal downtime due to these errors. In cases where the operation of a new package revision is questionable, we install it first in the experimental tree */local/new*. When it is verified as working, we install the new version in */local* and the old version in the archive tree */local/old*, so that users who still need the old version can access it. Users are told that new packages always appear first in */local/new* and old packages persist in */local/old*, so there is minimal disruption of service even if a new package fails.

Advanced Usage

Of course, the description of our policy and SLINK above is quite oversimplified and suitable mainly for novices. In practice, however, few repository maintainers must deal with more complex issues, and these issues are almost always issues of policy rather than tool use.

One pressing problem in installing advanced software packages is to insure that users have proper settings for environment variables. Solving this problem is simple provided that policy clearly indicates a course of action. Our policy is that */local/env* contains files of the form *package.cshrc*, *package.profile*, etc., for each installed package and relevant shell. When the user invokes a shell, all startup files matching that shell in */local/env* are sourced by the system shell startup file. These startup files are general shell scripts that can perform a variety of functions, including defining aliases and setting environment variables.

Admittedly, this is an imperfect solution. Mistakes in a startup file can disable a shell, and any student with access to */local/env* can execute an arbitrary command as any user. So trust between students and administrators is again essential, and */local/env* must be constantly monitored for malicious changes.

A second problem that plagues all repository management tools is that inter-package dependencies make it difficult to separate interdependent software tools into distinct packages. A classic example of this is the emacs info tree, to which all gnu applications contribute, but which has a unique index in a separate package from any of the applications. Our policy, where possible, is to separate such trees into their own packages so that they will outlive any contributing package. In the case of info, we utilize the *image*

/local/info as the actual info tree, and create the index as a file in that tree. This is terrible style, but no object-oriented approach can do better.

Libraries for Developers

Our tools are not only provided in a Perl5 script 'slink', but are also included in a set of Perl5 library modules that can be utilized by other tool implementors. Creating this library was much more difficult than writing SLINK itself, because of the amount of specification required by library functions and the complex couplings that occur between them. The library has several general-purpose modules for distinct purposes, of which the most important are:

- *Duper.pm* – implement high-level filesystem assertions.
- *Logger.pm* – enable writing of log and error files to disk.
- *Mapper.pm* – remember a map of things created by SLINK.
- *Protector.pm* – read and interpret virtual protections.

To utilize the library within a Perl5 program, one must create an object instance of *Slink::Duper* by a somewhat involved procedure:

```
use Slink::Logger;
use Slink::Mapper;
use Slink::Protector;
use Slink::Duper;

$logger = new Slink::Logger ({});
$mapper = new Slink::Mapper (
    'logger' => $logger
);
$protector = new Slink::Protector(
    'logger' => $logger
);
$duper = new Slink::Duper (
    'logger' => $logger,
    'mapper' => $mapper,
    'protector' => $protector,
);
```

After this quite complex initialization, with many other options not listed due to space requirements, SLINK's functions are available as the following library functions, where *\$source* is a source pathname and *@images* is a Perl5 array of images to create:

```
$duper->link($source,@images);
$duper->unlink($source,@images);
$duper->copy($source,@images);
$duper->uncopy($source,@images);
$duper->destroy(@images);
```

The Future

Several ongoing enhancements are under development for SLINK.

SLINK's major deficiency is that it can not check the consistency of a configuration before implementing it. Each SLINK command is incremental, so that its effect is dependent upon existing conditions that may be created by preceding commands. Thus, the only way to reliably check a sequence of commands for consistency is to implement them. A safe way of checking a configuration is to create a memory representation of the machine's filesystem and have SLINK work upon that image rather than the real filesystem in order to check consistency. This would provide a dry-run mode for SLINK that would inform users of what SLINK would do if invoked. This is difficult, however, because of the lack of truly portable mechanisms for manipulating UNIX filesystems in Perl, and because of subtle differences in the function of particular filesystem commands in different varieties of UNIX. NFS semantics are particularly difficult to infer without trying to modify an NFS-mounted filesystem. We currently think it unlikely that a perfect, portable simulation will ever be achieved.

Root access is too powerful and needs to be controlled by informing root users of the consequences of their acts, or even by prohibiting dangerous acts. In solving this major problem, SLINK's virtual protection scheme is only a rough beginning. As systems become more complex and file sources more varied, we need a mechanism whereby vendors and administrators alike can document the disposition of files so that administrators and tools will not make mistakes in modifying those files. SLINK's protections are the bare minimum needed for our maintenance policy to work; I foresee many more, for uses beyond our site and needs:

- *rdist* – this file is distributed via rdist.
- *vendor* – this file should only be modified by the vendor.
- *from <host>* – this file was generated on the given host.
- *created-by <command>* – this file can be generated by running the given command.
- *depends-upon <path>* – this file depends upon the given one for its contents.

Several times I have learned the need for the *rdist* protection, by changing files that were later overwritten by *rdist* automatically! It would have helped greatly if I was informed, at the time, that I was doing something foolish.

Conclusions

While we are quite happy with our community of apprentices and admins and the educational experience the community provides, there are many limits to using a community for system and network maintenance. A community is a rapidly changing constituency, where students become involved, become uninterested, and are hired by industry to run networks (even before graduating!) without warning. As the primary goal of an advocate is to control her or his own

environment, documentation is nonexistent. This in turn means that an ideal task for an advocate is an installation that will be obsolete soon, so that another advocate can start over and redo the work each time a release becomes available. It is thus essential to have a full-time system administrator for the purpose of providing a sense of continuity in this rapid-turnover environment.

Also, we have admittedly made a conscious decision to favor productive output over system consistency. This decision is not for everyone, and would not work in production environments, though we have fared well. Serious concerns over security, integrity, and trust of student workers invalidate our whole management philosophy when sensitive information is at risk. In these cases a volunteer is nothing less than a security risk.

Rather than being diminished in importance, the full-time administrator must perform more roles: mentor, manager, and auditor. The result, however, is that much more can be accomplished than one full-time administrator could possibly do without a backup of a community of helpers. Another very positive side-effect is that users who need something very often stop complaining and start taking responsibility for everyone's environment. This leads to a general appreciation for the job of administrator and the difficulties and skills it encompasses. And this, in my view, advances the profession from skill to art.

Acknowledgements

I wish to thank the many people whose thoughtful input and tolerance improved this software and policy. David Krumme was a constant supporter and advocate of the software and endured many brainstorming sessions to hammer out the truth from rough ore. Greg Owen was instrumental in implementing the initial algorithms. Grant Taylor forced me to put my policy in writing, and his thoughtful arguments over several months convinced both of us that we were moving in the correct direction. George Preble, Chris Leduc, Jonathan Rozes, Allan Stratton, and many others endured the initial bugs in trying out new features.

Availability

SLINK is freely available from <ftp://ftp.cs.tufts.edu/pub/slink>. The version described herein is 5.0.2.

Author Information

Alva L. Couch was born in Winston-Salem, North Carolina where he attended the North Carolina School of the Arts as a high school major in bassoon and contrabassoon performance. He received an S.B. in Architecture from M.I.T. in 1978, after which he worked for four years as a systems analyst and administrator at Harvard Medical School. Returning to school, he received an M.S. in Mathematics from Tufts in 1987, and a Ph.D. in Mathematics from Tufts

in 1988. He became a member of the faculty of Tufts Department of Computer Science in the fall of 1988, and is currently an Associate Professor of Electrical Engineering and Computer Science at Tufts. In 1996 he received the Leibner Award for excellence in teaching and advising from Tufts. He has assisted in maintaining the Tufts computer systems for Computer Science teaching and research since 1985, when he was a Ph.D. student, and is currently responsible for maintaining the largest independent departmental computer network at Tufts. He can be reached by surface mail at the Department of Electrical Engineering and Computer Science, 161 College Avenue, Tufts University, Medford, MA 02155. He can be reached via electronic mail as couch@cs.tufts.edu. His work phone is (617)627-3674.

References

- [1] Jonathan Abbey, "opt_depot web site", http://www.arlut.utexas.edu/csd/opt_depot/opt_depot.html.
- [2] David Bianco, Travis Priest, and David Corder, "Cicero: a Package Installation System for an Integrated Computing Environment" <http://ice-www.larc.nasa.gov/ICE/doc/Cicero/cicero.html>.
- [3] Wallace Colyer and Walter Wong, "Depot: a Tool for Managing Software Environments", *Proc. LISA-VI*, 1992.
- [4] Michael Cooper, "Overhauling Rdist for the '90's", *Proc. LISA-VI*, 1992.
- [5] Alva Couch and Greg Owen, "Managing Large Software Repositories with SLINK", *Proc. SANS-95*, 1995.
- [6] Alva Couch, *SLINK Manual*, 1996. <http://www.cs.tufts.edu/~couch/slink.html>
- [7] Tim Hunter and Scott Watanabe, "Guerrilla System Administration: Scaling Small Group Systems Administration To a Larger Installed Base" *Proc. LISA-VII*, 1993.
- [8] Kenneth Manheimer, Barry Warsaw, Stephen Clark, and Walter Rowe, "The Depot: a Framework for Sharing Software Installation Across Organizational and UNIX platform boundaries", *Proc. LISA-IV*, 1990.
- [9] Arch Mott, "Link Globally, Act Locally: A Centrally Maintained Database of Symlinks" *Proc. LISA-V*, 1991.
- [10] John P. Rouillard and Richard B. Martin, "Depot-Lite: A Mechanism for Managing Software" *Proc. LISA-VIII*, 1994.
- [11] John Sellens, "Software Maintenance in a Campus Environment: the Xhier Approach", *Proc. LISA-V*, 1991.
- [12] Walter C. Wong, "Local Disk Depot - Customizing the Software Environment" *Proc. LISA-VII*, 1993.

Managing and Distributing Application Software.

Ph. Defert, E. Fernandez, M. Goossens, O. Le Moigne, A. Peyrat, I. Reguero
– CERN, European Laboratory for Particle Physics

ABSTRACT

The paper describes a project for distributing application software in the large worldwide High Energy Physics (HEP) community. Hundreds of packages are maintained centrally and users can access them directly through the network. Workstation administrators can optimise access performance and reliability by specifying in their installation scripts the packages to be copied locally or accessed remotely. Product maintainers have a set of tools to generate their packages from the sources, while site administrators can replicate (part of) the central file-base manually or automatically. The generation takes place in different physical network domains. Replication on remote domains implies only propagating changes of the repository.

Introduction

Nowadays, UNIX systems almost always come with the bare minimum of software and in most cases this is insufficient for program development or data analysis. Commercial software tends to be specific and often too expensive. Thanks to the ubiquitous Internet, many users, especially those working in the scientific and educational fields, can access a lot of excellent software, such as editors, window managers, document formatting utilities, drawing tools, etc., most of which are freely distributed in source form.

The ultimate goal of ASIS (the Application Software Installation Server) project is to offer users access to a large number of freely available UNIX applications in order to optimise their productivity and working environment. In our system, end-users do not need to obtain, configure, generate, install and manage the software themselves. Managing the collection of applications is performed centrally by a team of system engineers with dedicated tools to generate and install the packages.

The *repository* is the place where applications are stored in an executable format. The application domains presently covered by ASIS are very diverse: the CERN Program Library, the core of the High Energy Physics (HEP) data analysis software, most GNU packages, the latest TeX/LaTeX setup, MIT X-windows and contributions, TCL/TK based software, PERL based packages and many other tools written by the UNIX community. Presently, the system gives users access to some six hundred products for eight different platforms, i.e., the hardware of SUN, DEC, IBM, SGI and HP with their different UNIX flavours. The number and kind of supported platforms are constantly reviewed according to user demand and available resources. The repository contains a data base describing all packages with version control and support information. It also includes tools for users who want access to ASIS, for product maintainers to

perform their management tasks and for remote sites to replicate the environment locally.

A user gets access to the software on an ASIS Local Copy by running the *client* software. With a small addition to the local system setup file, the *ASISwsm* (ASIS WorkStation Manager) script will be executed for each change in the ASIS Local Repository. In this way, new versions will be visible on the local machine and all software package versions will remain in step. This part of the system is specially kept simple since it is used by all UNIX-based workstations at CERN.

A suite of tools helps *product maintainers* keep their packages up-to-date. These tools automate the generation and installation process, with quality and reliability receiving maximum attention. Version control and documentation are also given great care.

The *replication* software allows a remote site or a cluster at the parent site to replicate a master repository on its local infrastructure. The script *ASISlcm* (ASIS Local Copy Manager) is run each time the local administrator decides to update the local copy of ASIS.

The Repository

The structure of the repository was designed with great care and special attention was given to the maintainability and ease of use of the client software, management tools and replication procedures.

ASIS can handle multiple platforms and versions. This means that each product is available for all (if possible) supported platforms, and that several versions of the same product can be present simultaneously in the repository. The "Reference Copy" of ASIS is used to manage the different packages, but should never be used by end-users. "Local Copies" of ASIS are built using the "Reference Copy". Users access a "Local Copy" of the ASIS repository via a

distributed file system, independently of the type of file system on which it resides. For instance, at CERN, "Local Copies" of ASIS are available on several distributed file systems: NFS, the SUN Networked File System, AFS, the Andrew File System, and more recently DFS, the OSF/DCE Distributed File System. Each distributed file system has its own "Local Copy" of the "Reference Repository". Users never access the "Reference Copy" directly.

Heterogeneous Platforms

Because we are working in a heterogeneous environment, i.e., a combination of more than one architecture and operating system (O.S.), the software is partitioned into two parts:

- a *specific* part, for each O.S., containing binary files and libraries;
- a *shareable* part, usable on all platforms,

containing scripts, include files, fonts, startup files, man pages, etc.

Each *specific* part must contain all needed references to the *shared* part to make sure that the repository and the products are self contained. In order to avoid unwanted site dependencies, all symbolic links are relative and independent of the root directory of the project. This makes the software easily exportable.

Platform naming conventions in ASIS are based on the ones adopted by the GNU project and by AFS. The result of the GNU *config.guess* script is the concatenation of three strings, the chip name, the vendor and the operating system with its version number. AFS names are directly determined by the AFS project. For efficiency in AFS, our most-used distributed file system, and for historical reasons, the ASIS platform identifier is the AFS one. GNU names are also

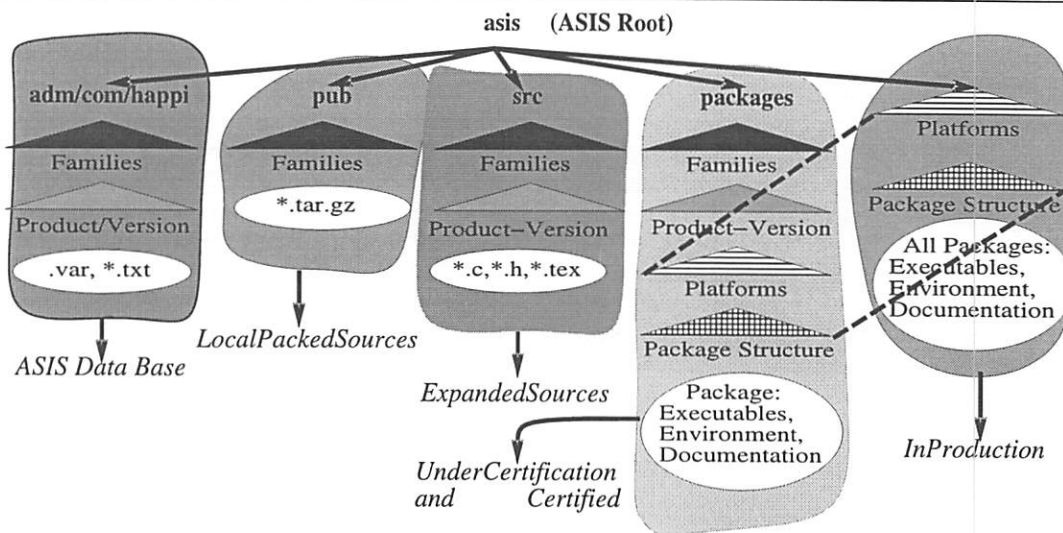


Figure 1: The Structure of the ASIS Reference Copy (or Repository)

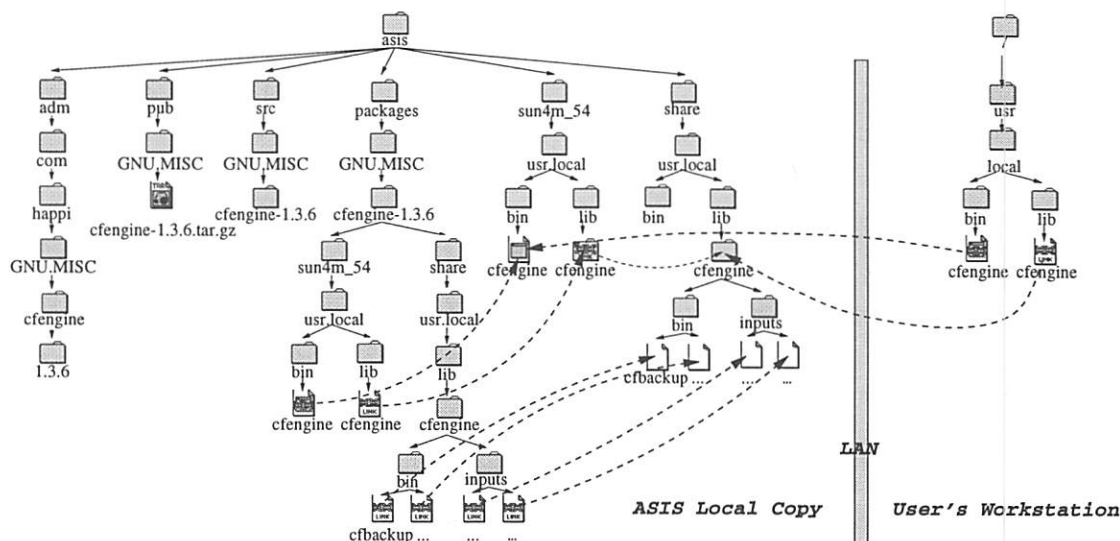


Figure 2: The GNU.MISC/cfengine-1.3.6 product in an ASIS Local Copy and on a User's Workstation

supported (see Table 1). The common part to all platforms is named *share*.

GNU name	ASIS/AFS name
Currently Supported	
sparc-sun-sunos4.1.3	sun4c_411
sparc-sun-solaris2.4	sun4m_54
mips-dec-ultrix4.3	pmax_ul43
hppa1.1-hp-hpux9.05	hp700_ux90
rs6000-ibm-aix3.2.5	rs_aix32
rs6000-ibm-aix4.1.4	rs_aix41
alpha-dec-osf3.2	alpha_osf32
mips-sgi-irix5.2	sgi_52
Planned Support	
hppa1.1-hp-hpux10.10	hp700_ux100
i386-unknown-linux207	i386_linux2

Table 1: Standard names for various platforms

Versioning

In the repository, multiple versions of a package can coexist. Users have access to versions which can be:

- *InProduction*: the currently supported version which is included by default in the user's environment;
- *Certified*: a version which is present in the repository, but made available to users only on request.

The 'Software Processing Model' section gives formal definitions of these two terms.

Generally, only one version is *InProduction*, i.e., formally supported but, exceptionally, more than one version of a software package can be put *InProduction* (for instance to ease transition to a new major release). For instance, presently, at CERN, several versions of *PERL/perl* and *TCL/tcl* are *InProduction*.

Certified products are stored in their own separate directory structure containing both *specific* and *shared* data, such as in the NIST Depot [9]. The "ASIS package format", shown in Figures 1 and 2, defines the following directory structure:

- `<family>/<name>-<version>` product name;
- `<platform>` identifier;
- execution directory as seen by the user. In our example, it is *usr.local* representing */usr/local*.¹
- The subdirectories needed by the particular package.

For reasons of convenience, products have been classified in different families like *GNU.EDIT/emacs*, *MISC/xemacs*, *X11.R5/fvwm* or *INET/pine*. The family can identify an origin and a function (*GNU.EDIT*), an application domain (*X11.R5*, *INET*), or a catch-all

¹Another possibility could be *usr.asis* for a system where the supplementary ASIS packages are installed below the */usr/asis* root directory. */cern* is the root for the CERN Program Library.

category (*MISC*). Search tools are available to find a product by specifying a regular expression for its name or one of its commands.

On the other hand, all products *InProduction* are stored in the same directory tree (see Figure 1). Files are placed by *specific* platform and *share* directly below the ASIS root directory, i.e.,

- `<platform>` identifier;
- execution environment directory (*usr.local*);
- all subdirectories needed, i.e., those required by at least one package *InProduction*.

The correspondence between files and products is registered in the ASIS data base. Only one physical copy of each product is kept in the repository. Thus, when a product is put *InProduction*, all files associated with the given product are moved from the *Certified* to the *InProduction* area.

ASIS Client Software

Users can have access to the contents of ASIS, its description and its documentation.

Access to the Data Base

The ASIS data base contains:

- the list of families with their description;
- the list of products with their description, including version-specific information;
- the state and history of transitions of all versions of a product;
- support information for each product, namely, the author's name and address, the local product maintainer's coordinates and the support level for the package; this support level goes from A to E depending on the availability of the local product maintainer to help the users and track bugs;
- the list of commands and files corresponding to each package;
- the description of how to install multiple versions of a particular product in the user environment, if applicable;
- the list of platforms supported in ASIS, active or frozen, O.S. support included or not, etc.

The script *ASISinfo* offers line mode access to the data base described above. Daily, an HTML copy of the data base is generated using the *ASISinfo* library. In addition, HTML pointers are created to the actual documentation distributed with the packages. Any WWW browser can be used to access these pages.

Access to the Applications

At CERN, as well as in most HEP centres, the UNIX support teams have chosen */usr/local* as the local root prefix for public domain applications (see [14]). One important requirement formulated by many users and some individuals was the possibility for their system administrator to be able install in */usr/local* software which is not contained in ASIS. As a consequence, ASIS has to maintain the directory tree

below */usr/local* on each client workstation in a very careful and coherent manner.

The *ASISwsm perl* script builds the directory(ies) where applications should reside (in this case */usr/local*) in such a way that all packages have the correct execution environment. It creates links from the directory */usr/local* to the distributed file system with the corresponding "Local Copy", for instance, at CERN, */afs/cern.ch/asis*, */nfs/cern.ch/asis* or *:/asis* (the DFS local cell). The script does not modify user files but reports possible file name conflicts between ASIS and the local environment.

A local system administrator can customise the behaviour of *ASISwsm*, specifying a set of options. These include:

- ignore a product;
- copy a product locally, instead of generating links to the distributed file system;
- generate links only for a product, even if the product maintainer forces a local copy;
- install a non-default version (i.e., not *InProduction*); this can be a specific version or the *Latest Certified* one;
- enable or disable overwriting user files.

This customisation is performed by modifying the local configuration file of *ASISwsm*. A line mode interface as well as a TK-based configuration editor are available (Figure 3 shows this TK-based product configuration window).

In addition, *ASISwsm* also allows the product maintainers to assign a different access type or to define post-installation commands. For instance, one can force a local copy for login shells like *GNU.SYS/bash* or *MISC/tcsh* or, it might be necessary on some

platforms to give the root set-uid bit for the command *xterm* of the *X11.R5/mit* product.

The ASIS repository evolves rapidly as products are continuously updated, bugs are corrected or new packages are introduced. Thus, *ASISwsm* should be run on a user's workstation at the same frequency as the corresponding "Local Copy" is updated. Anyhow, it can be run more frequently without harm since it only uses minimal system resources. If *ASISwsm* detects that the local copy of ASIS has not changed since its last run, it exits smoothly without doing any work. Moreover, *ASISwsm* is an idempotent command, i.e., its result is identical if it is fully executed once or multiple times. In other words, interruptions can be recovered by re-running the command.

Most users install the default environment *InProduction*. As all files of *InProduction* packages reside in the same "file system" or "volume" and "directory tree", *ASISwsm* can optimise the number of links to be created in the execution directory. For instance, if all links to be created in a directory point to the same target directory, then a link is created at the directory level rather than creating links for each file.

Similarly, to improve performance and reliability, replicas are created for the most frequently accessed file systems, i.e., the more common platforms. Replication is possible with sophisticated automounters in the NFS world and is part of the distributed file system itself, like in AFS and DFS.

As a side-effect, the way that the *InProduction* area was designed allows users to get access to ASIS by creating simple links pointing from the directory */usr/local* to the *InProduction* area. Note that in this

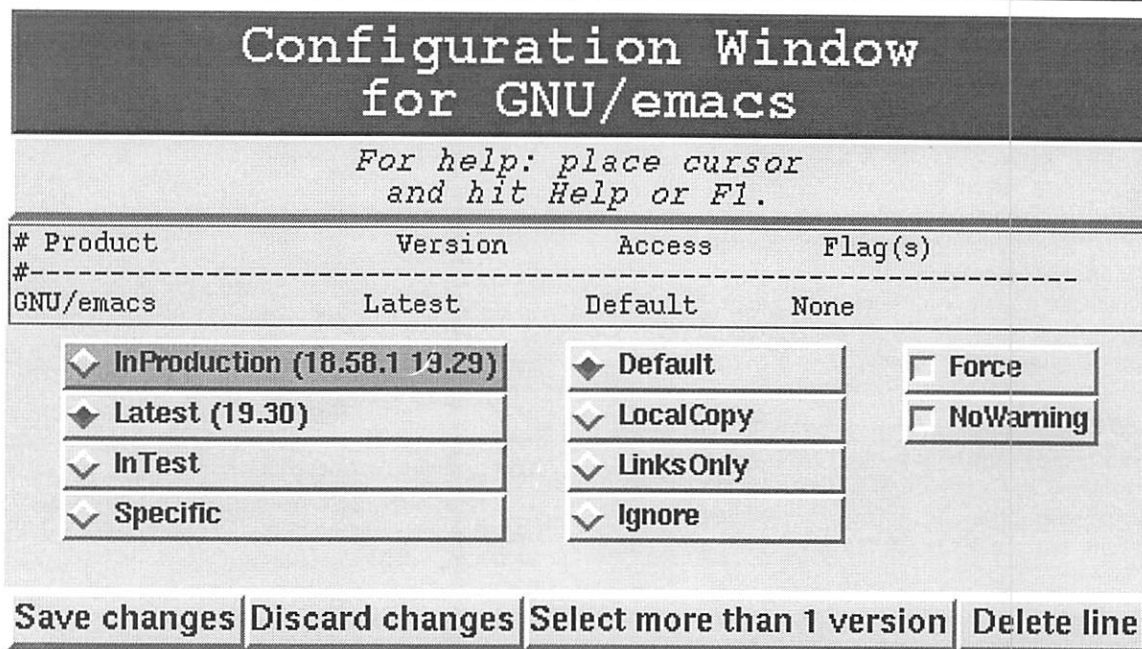


Figure 3: The "product configuration" window

case it is impossible to install anything else in */usr/local* or to use other versions than those *InProduction*.

Per User Environment Customisation

On a user-based system, different users may have different requirements as to which version of a package they need or want. On the other hand, the script *ASISwsm* customises software access at the level of the workstation. Therefore, it was necessary to introduce an ASIS tool to customise a user's environment. This tool is quite similar to the *setup* command in the UPS/UPD project at Fermi-Lab (see [2]).

The command *ASISsetup* is called by a user to change the environment to get access to a version of a product different from the default on the local system. Unfortunately, this procedure is not always possible. Some commands cannot be customised using shell environment variables. Moreover, some product maintainers cannot devote enough time to enter this setup definition in the ASIS data base.

The Repository Management Model and Software

The Software Processing Model

For over a year, we have installed public domain packages manually. During that period, we were able to observe the various steps that packages go through, from the moment that the sources are released until when the programs are delivered to the users. However, since the number of packages to be installed or updated constantly grew, as did the number of supported platforms, it soon became clear that the procedure had to be automated and that from the very start scalability and version control had to be taken into account.

Thus, based on our observations during the manual installation phase, we adopted for ASIS a "Software Processing Model" represented as a state/transition diagram in Figure 4. The model defines the following states:

- *Unknown*: the package is not present in the data base.
- *RemotePackedSources*: the system knows from which remote site it can get the source and where to store it.
- *LocalPackedSources*: the sources as distributed by the author(s) are stored in the repository.
- *ExpandedSources*: ASCII sources are available in the repository.
- *ConfiguredSources*: sources are ready to be compiled on each platform, generally after generating Makefile(s) and configuration file(s).
- *Executable*: the executable files were built.
- *Tested*: the tests provided with the package distribution kit were run successfully.
- *Installed*: the executable files, their execution environment and all available documentation were copied into a single scratch directory and checked for consistency.

- *UnderCertification*: the ready-to-use files and the full documentation were copied into the ASIS reference repository, where they are available for "testers".

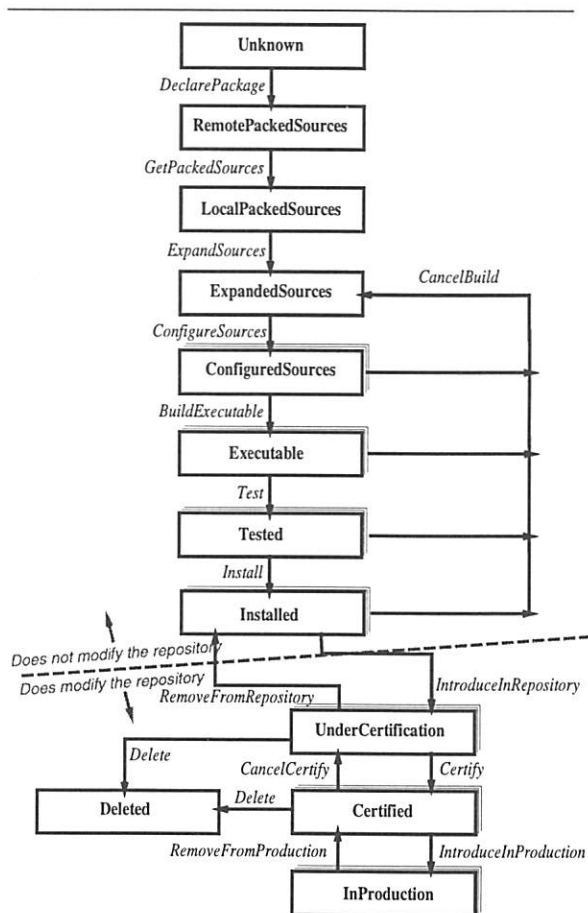


Figure 4: The "Software Processing Model"

- *Certified*: the software is available to all users after validation by the "testers", the manager of the local copy has updated the local repository.
- *InProduction*: the software is included in the default user environment and is officially supported.
- *Deleted*: the product is removed from the repository.

All states from *ConfiguredSources* to *InProduction*, are specific to each supported platform.

These states together with all possible state transitions (shown in *italic*) are seen in Figure 4.

The state transitions can be grouped according to their effect on the repository and can be

- *passive*, when they leave the reference repository unchanged. This is the case for all transitions above the dashed line in Figure 4, i.e., between the states *Unknown* and *Installed*.
- *active*, when they modify the reference copy of ASIS. These are transitions which cross or are situated below the dashed line of Figure 4, i.e., all transitions between the states *Installed* and *InProduction/Deleted*.

It is seen that passive transitions can be executed completely independently on different products while active transitions act on the unique reference repository. Therefore the execution of active transitions must be strictly controlled.

The result of the full sequence of passive transitions for a particular product is an object containing the full package, its environment and its documentation for all supported platforms in "ASIS package format" (see Figures 1 and 2). The transition *IntroduceInRepository* moves the object into the repository. All other active transitions manipulate the object.

It is also possible for a product which is distributed in executable form to be brought directly into the repository without going through any of the passive transitions. This is acceptable provided that the ASIS object to be introduced into the repository has the right structure. For instance, such procedures are applied for products like *MISC/xemacs* or *X11.R5/netcape*. Often, but not always, the process to build the right structure for such binary distributed packages can be automated. The *Install* transition creates the right directories and copies the given binaries into the right position; all other passive transitions are no-operation procedures.

The Automated Generation Environment

A product maintainer can perform all *passive* transitions (see above) in an automated way using a tool called *happi*, the "Heterogeneous Automated Product Processor and Installer". *happi* can execute a single transition, change the state of a product, or display the current state together with associated internal information. *tkhappi* is a TK-based user-friendly interface of *happi*, which improves also the readability of its output (see Figure 5).

General tasks, like getting the original distribution of a package or expanding a tar file, are performed on a server computer. For each supported platform, there exists a dedicated computer, where executables are built. The computers service these generation tasks and are called in ASIS jargon the "reference machines". *happi* is responsible for distributing the operational tasks for the software processing to these reference machines. *happi* performs the authorisation and the control of the remote processes. It is written in such a way that the tasks executed on the remote computers just execute *happi*. When *happi* is run, it first detects which tasks have to be run and where. Then, it launches as many concurrent tasks as possible. General tasks are mainly sequential, but O.S.-specific generations are executed simultaneously on all the reference machines. *happi* verifies the status returned by each task and determines the next actions to undertake. It also updates the data base describing the new state reached on each architecture.

happi collects logging information and builds log files and a history for each product. Logs and history can be most conveniently consulted via *tkhappi*, but any other text viewer can also be used.

happi only deals directly with passive transitions, hence several *happi* tasks acting on different products can run at the same time.

Task communications are handled mainly using *expect* [8], while all other functionalities are programmed in *perl* [15].

The "product maintainer" must provide *happi* with a "configuration file", written in *perl*, where all passive transitions are described. Each transition is represented by a *perl* function whose name is identical to the name of the transition (Figure 4). In addition,

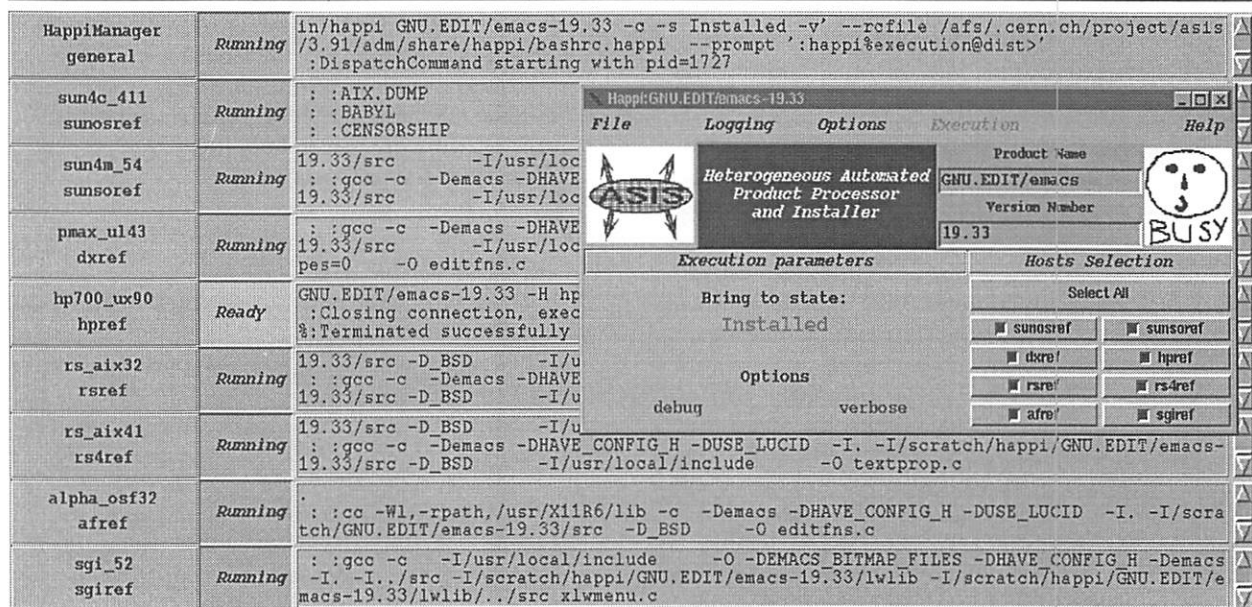


Figure 5: *tkhappi*: the tk based interface to *happi*

other parameters are represented by *perl* variables and can be assigned a value, overwriting the defaults if needed. Each function contains the sequence of operations to be executed in order to perform a successful state transition. In most cases, this file is very simple, since many packages are well designed and easy to build. Moreover, *happi* comes with an extensive library of functions for the most-frequently performed operations. Figure 6 shows a typical configuration file. Figure 7 shows a more complex case. In the latter, some work had to be devoted to correctly separate the installation of the *specific* and *shared* files.

happi is delivered with libraries, called packages in the *perl* jargon, that cover most of the repetitive operations:

- the standard *happi* library, containing dedicated *happi* specific operations;
- the *GNU library*, specific to the GNU products;
- the *documentation generation library*, to build Postscript and HTML files from the package documentation written in LaTeX or *texinfo*;
- the *file manipulation library*, operates on files,
- the *synchronisation library*, to synchronise the tasks on the different reference machines;
- the *O.S. knowledge base library*, to find optional libraries, include files, optimal compilation/link flags, etc. for the different supported platforms;
- the *ASIS data base query library*, to get information on other products in ASIS.

Examples of use of *happi* packages are shown in Figures 6 and 7. Some of the subroutines are described below.

- *Split* runs *GNU tar* to expand the sources. It uses standard locations both for the input, the compressed tar file and for the output, the ASCII sources.
- *Run* executes a shell command. It performs logging and error detection, analysing both the return value of the command and its output.
- *Link* runs the *X11 Indir* command on the local reference machine in order to create links to the centrally stored sources. It is the standard way to get access to the sources on the different reference machines.
- *Make* runs *GNU make*.
- *ModifyFile* edits a file.
- *DefaultExecDir* builds the user execution directory name for the given keyword.
- *DefaultInstallDir* returns the installation directory name according to the "ASIS package format" for the given keyword.
- *Sharelink* creates a link pointing from the *specific* part of a package to the corresponding *shared* area.
- *gnu::MakePSandHTML* builds Postscript files from existing *dvi* files and HTML files from the corresponding *texinfo* files. This is targeted to GNU software: it assumes that documentation

is written in the *texinfo* format. In fact, the *gnu::MakePSandHTML* function is just another name for *doc::MakePSandHTMLFromTeXInfo* which belongs to the documentation package.

- *file::copy* copies a file.

- *file::wipeout* deletes recursively directories.

In Figures 6 and 7, also some *happi*-specific variables were used:

- *\$final_automatic_state*: the default state to which *happi* brings a product if not specified otherwise;
- *\$packedfile*: the relative position of the product file in the *LocalPackedSources* area; this variable is automatically assigned to a standard value at *happi* startup; it may be overwritten by the product maintainer if necessary;
- *\$sys*: the standard platform name (read-only);
- *\$share*: a boolean which is true on the reference machine dedicated to do the work for *share*, for instance to generate Postscript and HTML documentation.

Other variables are available to deal with complex generation and installation procedures.

As soon as the product maintainer has provided the system with the *happi* configuration file, the system can run completely automatically. The command *mirror*² is run periodically to interrogate all distribution sites about their contents and to retrieve new sources for the packages supported in ASIS. *mirror* stores the retrieved files in the ASIS repository. Similarly, local authors can drop new distribution kits in the ASIS *LocalPackedSources* area.

Independently from *mirror*, *ASISdntv* (ASIS Detect New Versions) is run regularly (or executed manually, if needed). It scans *LocalPackedSources* and detects new versions of packages supported in ASIS. *ASISdntv* automatically launches the execution of *happi* which brings the product in the state specified by the ASIS parameter *\$final_automatic_state* set in the *perl* configuration file. Thereafter, *ASISdntv* sends a report to the product maintainer describing the result of the execution of *happi*. The product maintainer can read the *happi* execution logs with *tkhappi*. Even when the execution was diagnosed to be successful, the generation logs should nevertheless be verified to ensure that the process was actually correct since the available expert system for log analysis is still rudimentary. When the generation appears to have failed, *tkhappi* is used to help fix the problems.

The ASIS Transaction Model

Even though the full Software Processing Model describes the actual operations acting on a package satisfactorily, active transitions cannot be performed

²*mirror* was written by Lee McLoughlin (lmjm@doc.ic.ac.uk) and is a package written in Perl that uses the ftp protocol to duplicate a directory hierarchy between the machine it is run on and a remote host.

```

### -*- mode:perl -*-
#       File for X11.R5/tgif
$final_automatic_state = 'Installed';
# The tar.gz file is not in the standard pub/X11.
# It is in a subdirectory of pub/X11, given just below.
# Mirroring the whole ftp.x.org:/contrib directory
# gives us no control on the contrib directory tree
# structure.
$packedfile=
  "contrib/applications/$name/$name-$version.tar.gz";

sub ExpandSources {
# Run tar to extract the sources
  &Split;
}

sub ConfigureSources {
# Link to centrally stored sources
  &Link;
# Generate makefiles
  &Run('xmkmf');
  &Make('Makefiles');
}

sub BuildExecutable {
# The author defines TGIFDIR in a non-standard
# way. Below is the ASIS choice.
  &Make('TGIFDIR='.&DefaultExecDir('USRLIBDIR').'/tgif');
}

sub Install {
# /usr/local/lib/X11/tgif only contains shared
# data, thus we make a link from the specific
# /usr/local/lib/X11/tgif to the shared one.
  $tgifdir=&DefaultInstallDir('USRLIBDIR').'/tgif';
  &Sharelink($tgifdir);
# Install in 'Installed' area not in /usr/local/...
  &Make
    ('BINDIR='.&DefaultInstallDir('BINDIR').\
     "TGIFDIR=$tgifdir MKDIRHIER=mkdirhier install");
# Install man pages only on the machine
# dedicated to share.
  if ($share) {
    &Make
      ('MANDIR='.&DefaultInstallDir('MANDIR')\
       MKDIRHIER=mkdirhier install.man");
  }
}

```

Figure 6: The *X11.R5/tgif* configuration for *happi*. Comments were added for each step to aid comprehension.

```

### -*- mode:perl -*-
#           File for GNU.EDIT/emacs
$final_automatic_state = 'Installed';

sub ExpandSources {
    &Split;
}

sub ConfigureSources {
# On alpha's, use the OSF X11 libraries as no MIT X11 release 5. Better use
# cc then.
    &Link;
    $cfg_cmd='./configure --prefix='.&DefaultExecDir('prefix').
                '--with-x-toolkit=yes';
    $cfg_cmd .= ' --with-gcc=no' if ($sys =~/^alpha_osf/);
    &Run("$cfg_cmd");
}

sub BuildExecutable {
# Overwrite CFLAGS for performance reasons and change some bitmaps directories
    &Make('CFLAGS=-O bitmapdir='.&DefaultExecDir('bitmaps').':'.
                &DefaultExecDir('pixmap'));
}

sub Test {
    &Make ('check');
}

# Load GNU software specific library
require 'happi/GNUUtils.pl';

sub Install {
# Modify Makefile once for installing only the files specific to each platform;
# thereafter do the install in the "Installed" area
    &ModifyFile('Makefile');
    &Make("prefix=&DefaultInstallDir('prefix') install");
# Delete unnecessary created specific directories
    &file::wipeout(&DefaultInstallDir('datadir',$sys));
    &file::wipeout(&DefaultInstallDir('sharedstatedir'));
    &file::wipeout(&DefaultInstallDir('mandir'));
# create a link from specific to shared area
    &Sharelink("$dir{Installed}/$sys/usr.local/info",'F');
    if ($share) {
        *changed = *share_changed; &ModifyFile('Makefile');
        &Make("prefix=$dir{Installed}/share/usr.local install-arch-indep");
# Delete unnecessary created shared directories
        &file::wipeout(&DefaultInstallDir('libexecdir','share'));
        &file::wipeout(&DefaultInstallDir('bindir','share'));
        &file::unlink(&DefaultInstallDir('infodir').'/dir');
        &Makedir(&DefaultInstallDir('sharedstatedir').'/$name');
        &file::rmdir(&DefaultInstallDir('sharedstatedir').'/$name/lock');
        &file::symlink('/tmp',&DefaultInstallDir('sharedstatedir').'/$name/lock','O');
    }
# Install documentation (on share of course !)
# - Create the dvi files (nice we are with GNU)
# - Generate ps, HTML and Install in the 'doc' standard area
    if ($share) {
        &Make('dvi');
        &gnu::MakePSandHTML();
    }
}

# Subs to modify makefiles to separate share and the rest
sub changed {

```



```

s#install-arch-indep## if (/^install:/);
}
sub share_changed {
  s#$(srcdir)##$dir(ExpandedSources)#g if (/^COPYDIR\s*=/);
}

```

Figure 7: The GNU.EDIT/emacs configuration file for *happi*

without introducing some further control. Concurrency cannot be permitted if the system is to be kept simple and transitions are better performed in a coordinated manner.

For instance, when a product maintainer has to change the version of a product that is *InProduction*, two state transitions must take place: (1) a version is first *RemovedFromProduction*, and (2) a new one is *IntroducedInProduction*. Both these operations should be executed in the right order and both must be successful in order to ensure that the user environment is not modified drastically by leaving a product in an intermediate state, which prevents its further use after the next update. Moreover, if a product was only removed from production and no other version was introduced, users will no longer be able to access it.

It can happen that different products claim to be the owner of the same file. This can occur by bad coordination or wrong reorganisation of packages. If the repository has to remain coherent, only one modification can happen at any given time in order to guarantee that the introduction of a package does not destroy the execution environment of another. Thus, sequencing the different repository modifications becomes a necessity.

Therefore, it was necessary to introduce a transaction system for ASIS to ensure sequentiality and to introduce atomicity. Each transaction consists of a sequence of operations (transitions and/or specific actions). Each transaction is either performed completely and successfully or not at all. Transactions are

ordered and performed one at a time.

A product maintainer is thus guaranteed that the package being manipulated does not disappear from the *InProduction* area when actually trying to update the version *InProduction*, even if a problem occurs during the execution of one of the state transitions. Transactions are themselves executed sequentially which simplifies the control of the file name space and increases the file system integrity.

The ASIS Transaction System

The system is based on the cooperation between the ASIS Central Transaction Manager (*ASISctm*) and the Local Copy Managers (*ASISlcm*).

The Central Transaction Manager is responsible for checking, committing and distributing the transactions to be executed.

The Local Copy Managers are responsible for maintaining their own repository and, possibly, checking the feasibility of a transaction in the local environment.

A local copy is said to be *compulsory*, if any modification must be applicable to that particular copy as soon as this modification is valid on the reference copy. A compulsory copy can be updated from the reference copy at any time. Other copies are called *independent*.

A transaction is submitted by the product maintainer to the ASIS Central Transaction Manager which accepts or refuses it. The *ASISctm* program verifies that

File	Actions	Operations	Options	Help
ASIS Transaction Editor and Submitter				
Name	Upgrade to GNU.EDIT/emacs-19.32 and related stuff			
Description	Coordinated upgrade to GNU.EDIT/emacs-19.32 plus - GNU.EDIT/vm-5.96beta - GNU.EDIT/calc-2.02d - GNU.EDIT/auctex-9.3c			
Operations	RemoveFromProduction GNU.EDIT/emacs-19.30 IntroduceInProduction GNU.EDIT/emacs-19.32			
New operation	ChangeVersionInProduction GNU.EDIT/vm 5.95beta 5.96beta			

Figure 8: The Transaction Editor and Submitter (Prototype).

- the submitter is duly authorised for each operation and product of the transaction;
- the product does not conflict with other products in the file name space;
- the needed resources are available (disk space, access, etc.) for the reference and each *compulsory* copy.

When all checks are successful, the transaction is committed. The product maintainer knows that the modification will be performed successfully on the reference repository and that the next update of the compulsory copies will also be successful and will include the submitted changes. Updates of *independent* copies are not guaranteed.

The update of the reference copy will happen automatically without further intervention by the product maintainer. It is performed asynchronously when the *ASISlcm* (the ASIS Local Copy Manager) of the reference copy is run. The frequency at which *ASISlcm* is run is determined by the ASIS Manager.

The Central Transaction Manager maintains:

- the ordered list of committed transactions;
- the list of dependencies between transactions;
- the lists of compulsory copies and their attributes;
- the authorisation data base;
- the *virtual state* of the repository, i.e., the image of the repository with all committed transactions performed.

Product maintainers can use *ASISStes* (the ASIS Transaction Editor and Submitter) to ease manipulating and submitting transactions, i.e., the changes to the repository needed to maintain the packages. *tktes* is the corresponding TK-based Transaction Editor and Submitter (Figure 9).

tktes enables product maintainers to:

- build the list of operations to perform in a single transaction, submit this transaction to the Central Transaction Manager and, finally, commit it if correct;
- list the committed transactions;
- view the *virtual state* of the repository, i.e., the state of the repository (file name space, disk space, etc.) after execution of all committed transactions.

The Replication Model and Software

Presently, the ASIS repository is being mirrored by many HEP sites. Even though mirroring is optimised by only transferring modified files (as described in [1]), a complete traversal of the whole ASIS directory tree is necessary. The operational cost is thus proportional to the size of the repository, i.e., several thousands of files.

If all transactions, i.e., repository modifications performed on the master copy since the last local update, are known, it is unnecessary to scan all ASIS files. Some transactions like *IntroduceInProduction* can be performed entirely locally as they just move files from the local *Certified* to the local *InProduction* areas. Only modified and newly created files are to be transferred through the network. No check is done on unchanged data. The time to update the local copy then becomes proportional to the size of the changes made to the ASIS repository, which is more acceptable.

The ASIS Local Copy Manager (*ASISlcm*) updates the content of the local repository from a designated *master* copy. It obtains the “performed transactions list” from the *master* and determines, since the time when it was last run, the sequence of transactions

File	Action	Info	Options	Help
ASIS Local Copy Manager				
TID	MasterCopy	MTID	Name	
1	cern:ctm	1	Log of happi transition	<input type="button" value="Info"/> <input type="button" value="Refresh"/>
2	cern:ctm	2	Log of happi transition	
3	cern:ctm	3	Log of happi transition	
4	cern:ctm	4	Log of happi transition	
5	cern:ctm	5	Log of happi transition	
Name	Log of happi transition		TID	2
Master copy	cern:ctm		TID in master	2
Description	Send GNU.EDIT/emacs-19.32 in the Repository for Certification			
Operations	IntroduceInRepository GNU.EDIT/emacs-19.32 -H hp700_ux90 ...			
Committed by	defert.@CERN.CH on Tuesday August 6, 96 at 11:20:43			

Figure 9: The Local Copy Manager (Prototype).

to be executed to update the local repository. It verifies the feasibility of the transactions, performs the necessary operations and maintains the list of already executed transactions.

ASISlcm can be customised to control the following:

- the frequency and the scheduling of the updates;
- the location of the *master* copy, i.e., the replica from where to take the transactions and data;
- a runtime filter to apply before performing transactions, e.g., rejecting, delaying, or modifying some transactions according to local rules, like ignoring some families, products or platforms.

For an *independent* copy, the choice of the master and the update frequency is completely open. However, when *ASISlcm* is run on a *compulsory* copy, the master must be the "Reference Copy". In this case, the transactions list is read from the "Reference Copy" and new files are copied from there also. *ASISlcm* can be used to update the "Reference Copy". Then, no master is used and new files are taken from the *Installed* scratch area with transactions read from the list of "Committed Transactions".

The TK based *tklcm* allows a local copy administrator to view and manipulate the transactions list (see Figure 9).

The Status of the Project

Functionality

ASISwsm is now part of the standard CERN installation procedure of all UNIX systems and is used on several hundreds of workstations.

There are three ASIS local copies at CERN, to serve NFS, AFS and DFS users.

happi and *tkhappi* are used to maintain most public domain and CERN packages. At present, when a new source distribution kit is released at a distribution site, the full generation is not started automatically. The *ASISdsv* program does not launch *happi*: it just scans the *LocalPackedSources* area of the ASIS repository to find new versions of products and builds a "to-do" list for the product maintainers. Additionally, the product maintainer are informed by an electronic mail so that they can run *tkhappi*. All transitions in the "Software Processing Model" were implemented using idempotent functions, except *IntroduceInProduction* and *RemoveFromProduction* which could only be built as atomic operations secured by a check-point mechanism.

Beta versions of *ASISctm*, *ASISstes* and *tktes* run in the ASIS developers environment, but are not yet delivered to other product maintainers. The latter use a version of *tkhappi* which inserts an entry in the transaction list for each active transition performed, see Figure 9.

ASISlcm and *tklcm*, also in beta test, are used to manage pilot replicas for small clusters at CERN. By the end of 1996, the University of Barcelona should use *ASISlcm* to replicate the ASIS environment. Presently, ASIS is mirrored in many HEP sites like the Rutherford Appleton Laboratory (U.K.), CASPUR (INFN, Italy), etc. These sites, however, still use full mirroring. *ASISlcm* will be gradually introduced as soon as it has been thoroughly tested and validated.

Usage

At CERN, there are presently around seven hundred Sun's and HP's workstations each, two hundred IBM's, three hundred and fifty DEC's and fifty SGI's, giving a total of about two thousand UNIX workstations and servers. Some seventeen hundred X-terminals are also registered. More than 5000 users have a UNIX login account. Most of the workstations access the Application Software Installation Server and most UNIX users use software contained in ASIS for their main development tasks.

Around six hundred public domain packages are available for eight different platforms. Ten per cent of these products have more than one version *Certified* or *InProduction* in ASIS.

The size of the CERN ASIS local copy serving the NFS clients is about eleven gigabytes, forty per cent of which is devoted to the CERN Program Library. Similar copies are available for AFS and DFS.

The "Ecole Polytechnique Fédérale de Lausanne" (EPFL) now uses the ASIS tools suite to install public domain software and to manage the versions at their own site. They participated a lot to the debugging phase, mainly to resolve site-dependency problems. Recently, they have taken on the responsibility for the development and maintenance of the "User Interface" part, mainly based on TK. The EPFL group is getting more and more involved in the project and there is no doubt that future versions of ASIS will include features specified and designed by them.

Availability

The tools suite is in beta test, and during this period the sources are available from CERN. The CERN management has still to determine the policy for distributing production versions. All HEP centres are entitled to receive the software free of charge. Other academic institutes should hopefully get it also. In principle, others should contact CERN for more detail.

Other Information

The full documentation of ASIS is available at the URL: <http://www.wcn.cern.ch/dci/asis>. This gives access to the "ASIS Users and Reference Guide" [5], the "ASIS Product Maintainer's Guide" [4], and various related publications, plus the full set of documentation distributed with the packages included in the ASIS repository.

The full tools suite can be obtained from the URL <ftp://asisftp.cern.ch/dist/ASIS/ASIS.tar.gz>. The *tar* file contains a *README* file which serves as a very first prototype of the "ASIS Installation Guide". Currently, the distribution contains the version which is in production at CERN and does not contain the transaction system. The distribution will be regularly updated with bug corrections and improvements. The ASIS tools suite version 4 will be distributed as soon as it has been certified and the distribution policy defined.

Future Developments

Software interdependencies are at present not handled. We are studying possible solutions as proposed in Fermi-Lab's UPS/UPD [2], in Depot-Lite [11], or in the Debian Linux Distribution Project [7] which could be used as a basis for ASIS product dependency management.

Author Information

E. Fernandez Dominguez has completed his End of Degree Project at CERN in order to get a Telecommunications Engineer Degree at the Technical University of Madrid (UPM). He now works for "Telefonica Sistemas" in Madrid; contact him at [<eusebio@ete.tid.es>](mailto:eusebio@ete.tid.es).

O. Le Moigne has been working at CERN since he received his degree of Telecommunication Engineer of the ENSTB School (Brest, France) in 1995. He can be contacted at [<olivier.le_moigne@cern.ch>](mailto:olivier.le_moigne@cern.ch).

A. Peyrat received his degree of Computer Science Engineer of the ENSIMAG Engineering School of Grenoble (France) in 1993. He worked at CERN between 1993 and 1995. He can presently be contacted at [<a Peyrat@email.teaser.fr>](mailto:a Peyrat@email.teaser.fr).

I. Reguero received the "Ingeniero Superior de Telecomunicacion" degree from the Polytechnic University of Madrid in 1985. He studied in parallel two curriculum areas: "Data Communication / Data Transmission" and "Computer Science/Data Transmission". His graduation thesis was "Specification and Implementation of Interpersonal Messaging Protocol, Following the X.420 Standard". He worked in 1986 and 1987 as system engineer for the IBM IN network. He worked in 1988 for "Banco De Santander" as communications systems support of a network of more than 2000 offices. He is currently a systems engineer at CERN working on large systems performance and communications, network backup systems, Unix system software installation and consultancy for Unix system administrators. He can be reached at [<ignacio.reguero@cern.ch>](mailto:ignacio.reguero@cern.ch).

M. Goossens. After obtaining a MSc in physics in 1972 and PhD in physics in 1978 from the Free University of Brussels (Belgium) he joined CERN at the beginning of 1979. After working two years as a Fellow on a muon experiment, he moved to software

support working on memory management problems. Soon he came to realize the importance of good and up-to-date documentation, and thus became gradually more involved in the field of text processing and documentation. Over the years he worked with several typesetting systems, more in particular LaTeX and Framemaker, Postscript, SGML and HTML. Recently his main interest is the development of tools to facilitate using sources of documents for generating online hypertext documentation, like WWW and Acrobat. He is the co-author of a book *The LaTeX Companion* (Addison-Wesley 1994). In 1994, he was elected President of the French-speaking TeX Users' Group GUTenberg, and since 1995 also of the International TeX Users' Group TUG. He can be mailed at [<michel.goossens@cern.ch>](mailto:michel.goossens@cern.ch).

Ph. Defert received an MSc in Computer Science from the Free University of Brussels in 1975 and a PhD in Applied Mathematics in 1983 from The University of Namur (Belgium). He worked at the European Southern Observatory between 1984 and 1987 in Space Telescope European Coordinating Facilities in Garching bei Muenchen (Germany) as ST Data Analysis Scientist. After having moved to CERN to participate to the design and implementation of the control system of the Large Electron Positron Collider, he is now a member of the Computing and Networks Division of CERN specifically in the Distributed Computing Infrastructure Group. In this environment, he manages the ASIS Project, subject of this paper. His present research interests include Software Engineering. Contact him at [<philippe.defert@cern.ch>](mailto:philippe.defert@cern.ch).

References

- [1] Paul Anderson. Managing program binaries in a heterogeneous unix network. In *LISA V Proceedings*, pages 1-9. Usenix, 1991.
- [2] William Bliss, Jonathan Streets, Lourdu Udu-mula, and Margaret Votava. *UNIX Product Distribution, User's Guide*. Fermilab, FNAL, 1992.
- [3] Michel Dagenais, Stephane Boucher, Robert Gerin-Lajoie, Pierre Laplante, and Pierre Mailhot. Lude, a distributed software library. In *LISA VII Proceedings*, pages 25-32. Usenix, 1993.
- [4] Philippe Defert, Alain Peyrat, and Eusebio Fernandez Dominguez. *ASIS: Product Maintainer's Guide*. CERN, the European Laboratory for Particle Physics, 0.91 edition, 1995.
- [5] Philippe Defert, Alain Peyrat, and Ignacio Reguero. *ASIS: Users and Reference Guide*. CERN, the European Laboratory for Particle Physics, 3.00 edition, 1994.
- [6] Ph. Defert et al. Automated management of an heterogeneous distributed production environment. In *First Conference on Freely Redistributable Software*, pages 1-8, 1996.
- [7] Ray Dassen et al. *Debian GNU/Linux*. Debian, <http://www.debian.org>, 1996.

- [8] Don Libes. *Exploring Expect.* O'Reilly and Associates, Inc., 1995.
- [9] Kenneth Manheimer, Barry Warsaw, Stephen Clark, and Walter Rowe. The depot: A framework for sharing software installation across organizational and unix platform boundaries. In *LISA IV Proceedings.*, pages 37-46. Usenix, 1990.
- [10] John K. Ousterhout. *Tcl and the Tk Toolkit.* Addison Wesley, 1994.
- [11] John P. Rouillard and Richsrd B. Martin. Depot-lite: A mechanism for managing software. In *LISA VIII Proceedings*, pages 83-91, 1994.
- [12] John Sellens. Software maintenance in a campus environment: The xhier approach. In *LISA V Proceedings*, pages 21-44. Usenix,. 1991.
- [13] Stephen Shafer and Mary Thompson. The sup software upgrade protocol. Technical report, Carnegie Mellon University, School of Computer Science, 1988.
- [14] Rainer Toebbicke and Philippe Defert. Recommended standard for unix workstation environment setup. Technical report, CERN, the European Laboratory for Particle Physics, 1990.
- [15] Larry Wall and Randal Schwartz. *Programming perl.* O'Reilly and Associates, Inc., 1991.
- [16] Walter C Wong. Local Disk Depot – Customizing the Software Environment. In *LISA VII Proceedings*, pages 51-56. Usenix, 1993.

A New Twist on Teaching System Administration

Raven Tompkins – University of Indiana

ABSTRACT

The Unix Workstation Support Group (UWSG) is part of University Computing Services (UCS) at Indiana University. Although UWSG is a small, five member group, located on the main campus in Bloomington, the group provides advanced system administration consulting for the entire campus system. This includes eight branch campuses as well as the main campus.

As a growing number of departments on campus turn to Unix to meet their computing needs, the need for system administrators increases. Many departments lack the finances to both purchase a Unix workstation and hire a system administrator. For example, the UWSG set up a new departmental workstation in late 1995. By mid summer 1996, the department still did not have a system administrator and the systems main users had only a minimal knowledge of Unix. Compounding this problem is a general shortage of experienced system administrators on campus. This has resulted in an increase in the number of novice system administrators and Unix users interested in learning system administration. The UWSG has felt this increase directly, as group members continue to spend more and more time on basic system administration consultations.

Because many of these requests come from outside the campus system, it is apparent that this issue is not limited to the Indiana University computing environment. The number of universities offering system administration courses has increased. University departments who have access to these courses frequently cannot afford the time involved in enrolling a staff member. Additionally, system administration is often an upper-level course that also requires several prerequisite courses. The number of commercial Unix system administration courses has increased. However, the cost of these training courses is prohibitive for many who could use the instruction.

Project Development

In 1995 the UWSG was approached by the Computing Education group of UCS and asked to develop an Education Certification (EdCert) course for Unix workstation system administration. As the group began working on this project, it became apparent that a single course would not be sufficient. According to the SAGE paper, "SAGE Job Descriptions", <http://www.usenix.org/sage/jobs/jobs-descriptions.html>:

The best systems administrators are generalists: they can wire and repair cables, install new software, repair bugs, train users, offer tips for increased productivity across areas from word processing to CAD tools, evaluate new hardware and software, automate a myriad of mundane tasks, and increase work flow at their site.

Unix system administration encompasses a body of knowledge so vast that the sheer volume of material cannot be contained in one course. The SAGE "Job Descriptions" paper also states that a junior level system administrator should have one to three years

experience. System administration is an ongoing learning process. A training course can lay a foundation on which to build, but after the course, then what? Where does a novice system administrator go to learn more? Although there is a growing body of Unix documentation on the World Wide Web and in print. Here again, locating specific information can be time consuming and expensive.

During the many brainstorming sessions that gave birth to the Unix System Administration Independent Learning (USAIL) project, it became apparent that there were a number of basic system administration questions that UWSG members answered repeatedly. Group members were spending a substantial amount of time on the phone talking new system administrators through the same procedures and emailing out the same sets of instructions. Some of these issues could be addressed with existing online documentation, but there was no single Web site to which users can be pointed. In other instances the existing documentation was incomplete, too technical to be accessible to new system administrators, contained too much information, or lacked step by step instructions on how to complete a task.

Bearing in mind the limitations of a single training course, the UWSG decided on a more

encompassing approach that would incorporate the material developed for a Unix workstation system administration training course into a larger body of information for continued independent learning. The Unix System Administration Independent Learning (USAIL) project integrates documents written by the UWSG for its system administration course with existing Unix documentation and other system administration courses available on the Web. In order to take advantage of the growing body of online Unix documentation, USAIL needed to be Web based. This brought up a number of other issues. To whom does the information need to be accessible? How will the information be maintained? How should it look?

In order to extend the UWSG's ability to provide effective system administration consulting services, USAIL needed to be available to all nine Indiana University campuses across the state. This prompted discussion of whether USAIL access needed to be restricted to Indiana University users. The number of questions UWSG received from outside the campus system and the growing number of online system administration courses made it evident that the need for USAIL went beyond the IU campus. In the tradition of Unix as an open system, USAIL needed to be available to the whole Internet community.

There are other system administration courses available via the Web. The difference between USAIL and other courses use of the Web is twofold. Other courses use the Web to deliver information for use as supplemental material to a course textbook or books. Some courses publish only brief lecture notes or outlines. There are courses which make all the learning materials available, but only as the course progresses through them so they are not always accessible. All USAIL materials are freely available to anyone with Internet access. Using the Web as the delivery mechanism facilitates an open learning approach where users may work through the material at their own pace. The material may be started at any time, worked through as a course, or used as a reference to find answers to specific questions.

Because USAIL is entirely Web-based, the material can be updated, maintained and expanded in a timely manner. The Web also provides the means to make the material into a dynamic, interactive body of documentation, rather than a static one. Working through the course portion of USAIL requires active participation on the part of the user, whereas simply reading a book on system administration is a passive experience. For the first few months USAIL maintenance of hypertext links was done manually. Currently the UWSG is working to automate this process with a combination of existing link checker scripts and site-specific shell scripts. Optimally, USAIL links would be checked daily. Unfortunately, the time required to verify the output from the link check script makes this an impractical solution. The UWSG hopes to set up a system where links are checked weekly or bi-weekly.

In order to make USAIL accessible to as many people as possible it needed to be viewable from a variety of Web browsers. This required that the site not be graphics intensive. Although the recommended browser is the current version of Netscape Navigator, the project information may also be accessed with Lynx. Each USAIL page written by the UWSG contains a header, footer, and a series of navigation buttons. The header includes information on when the page was last updated. The footer contains the date, a small copyright notice, the URL of the document, and a "mailto" for contacting the UWSG.

Across the top of each page are four buttons: "contents", "contents+", "glossary", and "library". These buttons take the user to particular USAIL documents. The "contents" button is linked the main introductory page. The "contents+" button allows a user to view the expanded table of contents, which lists all the USAIL documents individually. The "glossary" button is linked to an online USAIL glossary. The "library" button is linked to the table of contents for the USAIL library.

Below these buttons are three small arrow buttons, that can be used to navigate through the project. The arrow buttons are also displayed at the bottom of each page. "alt" tags are used so that in a non-graphical browser the navigation buttons appear as text.

A USAIL style guide was developed to help create a consistent look for the project. The style guide includes typographical conventions, links to page templates, and guides for creating glossary entries and quiz data bases. Typographical conventions currently used in USAIL are based on common typographical conventions found in printed Unix system administration books. This means that physical HTML tags are being used rather than logical ones. Although there are logical tags specifically for user keyboard input, <KBD>, and variables that the user needs to replace with a value, <VAR>, most browsers render these the same way.

How USAIL Works

USAIL is a work in progress. At present it consists of three main parts: a self-study system administration course, glossary, and a reference library. The beginning knowledge level of the course was dictated by the services the the UWSG provides. Although the course begins with a Unix review section, a basic working knowledge of Unix is assumed. A "For First Timers" section includes links to tutorials for users interested in learning Unix.

The course is laid out under a series of subject headings, or topics, that encompass various aspects of system administration. Choosing topics was not an easy task. Much of the information that needed to be conveyed was interconnected. However, constructing a step by step training course meant that the information had to be laid out in some linear fashion.

The original list of topics came from a brainstorming session where different aspects of and tasks relating to system administration were written down on pieces of paper. The tasks were then grouped according to whether a system administrator must know how to do this, should know how to do this, or could know how to do this. From there the tasks were organized into topics. During development, the topic list has changed several times. The current list is as follows:

1. Unix Review
2. Basic Unix Concepts
3. Installing the Operating System
4. Integrating Your Machine with the Network
5. Administering Mail
6. Daily System Administration Tasks
7. Administering Peripherals
8. Backups
9. Administering Software
10. Performance and System Tuning

This list of topics is not meant to be an exhaustive list of all the information a system administrator needs to know. Topics are intended to be loose headings under which more specific information can be organized.

Each topic is subdivided into a series of documents, or modules, that pertain to that subject. The modules function as stand-alone references in addition to being part of a course of learning for prospective system administrators. In cases where the amount of information warranted it, topics are divided into subtopics and then into modules. This structure allows modules to be added as information and the needs of system administrators change. This is vital for keeping the course current.

While reviewing other online system administration courses on the Web and Computer Based Training software purchased by Indiana University, it became apparent that USAIL material should be presented in a dynamic, interactive manner. Long, unbroken segments of text are difficult to read through. This is particularly true if they contain technical or abstract conceptual information. With some Unix concepts, such as disk partitioning and filesystem layout, there is a divergence between physical and logical structure. Pictorial representations can be invaluable in learning how these concepts function. During several brainstorming sessions, where UWSG members discussed their own experiences with learning Unix and system administration, another important consideration arose. Reading information is not the same as learning it. In order to provide an opportunity for users to learn system administration, rather than simply read about it, part of the course had to be a hands on experience.

Graphics, lab exercises, auto-graded quizzes, and inline questions allow users working through the material to learn by direct participation. Most topics include lab exercises that are designed to provide hands on experience in working with the material. Many of the modules written by UWSG include inline

multiple choice questions mixed in with the text. The answers to these questions provide additional information on a given subject and an opportunity to begin "thinking in Unix". Most topics end with a quiz.

A Perl CGI script randomly selects quiz questions from data files and presents them as an online form. When the quiz form is submitted, it is graded by another Perl CGI script. A results page is then returned which tells the user whether or not they have passed and which questions were missed. Because the quiz questions are randomized, users may return to the same material for review and retake a quiz without receiving an identical set of questions each time. Even if a few of the questions are reused, they will not be presented in the same order. Questions are either multiple choice, or true/false. Each time a question is given, the order in which the possible answers are presented is also randomized.

One of the most recent additions to USAIL are three final, self-evaluation quizzes. Questions for these quizzes are selected from a master database that contains all the questions from previous USAIL quizzes. Each of the three quizzes is aimed at a different level of competence in beginning system administration. A quiz consists of sixty questions and can be completed in approximately forty-five minutes.

Thanks to Denis Howe's "Free On-line Dictionary of Computing", <http://wombat.doc.ic.ac.uk/>, the UWSG did not have to build the USAIL glossary from the ground up. The UWSG obtained permission to borrow and modify as many entries as needed, provided that credit was given to the original source. Currently the glossary mainly contains entries for terms that may not be familiar to beginning system administrators. The UWSG hopes to expand the glossary to include terms relevant to Unix in general. The glossary engine is also composed of a series of Perl scripts. In addition to displaying glossary entries, the scripts cross reference them and attach an attribution section which gives credit to the source of the definition. At the end of each module written by the UWSG there is a "Terms used" section. This lists the terms used in the module which may be found in the glossary. After several lengthy discussions the UWSG decided that terms should be noted in this manner rather than making them hot in the text itself. Most terms, once they are introduced, are used repeatedly throughout the modules. Making only the first use of a term hot would be inconsistent, but making the term hot each time it was used would be disruptive to the flow of text. The glossary can be accessed via a button at the top of all documents written by the UWSG. The glossary is constructed so that users may look up a particular term, view all the terms contained in the glossary, or look at all the terms that begin with a certain letter.

The third segment of USAIL, the reference library, is intended to serve as a central clearinghouse

for Unix documentation available on the Web. When system administrators need information on a particular subject, they can look for the appropriate topic in the library and find what they need. The library is organized into a series of library lists on general topics. Each library list includes an index to narrow the topic into smaller categories. Currently the library includes the following lists:

- HTML and the Web
- Networking
- Security
- X Windows
- Hardware
- Programming
- Online System Administration Courses
- Computer Humor
- Unix Software
- Unix Lore and History
- Backups
- Unix Flavors
- Misc.

The library is growing rapidly, with new lists or documents being added weekly. The introductory USAIL page includes a request for comments and suggestions. Suggestions and requests from users will allow the UWSG to expand the library to meet the user's needs.

Conclusion

USAIL constitutes a new approach to teaching system administration. This approach can fill the gap between short term training courses and actual practice, as well as provide an independent means of learning system administration. There are several advantages to this approach. As opposed to the high cost of commercial system administration courses, USAIL is freely available to any user who has access to the Internet. Because USAIL is Web-based, it does not rely on proprietary software or on a particular operating system. The amount of time spent on course material is completely up to the users, who are free to work at their own paces. If a system administrator needs information on a specific topic they can access what they need without having to perform an extended Web search to locate it.

USAIL material can be easily updated and expanded in accordance with user's needs. Using the Web as the delivery mechanism also allows the UWSG to take advantage of existing online Unix documentation and create documents where they are needed. In reviewing existing documentation UWSG members were surprised at what was and wasn't already available. For topics such as Unix security, there was an enormous amount of documentation. However, it was all focused on particular security issues or aimed at particular platforms. There seemed to be no general overview of Unix security that introduced the concepts involved or discussed the immediate security considerations involved in setting up a workstation. Similar issues arose in creating the

"Performance Monitoring and System Tuning" topic. There was very little documentation. Most of it dealt with kernel parameters, which is not a particularly accessible subject to novice system administrators. In creating USAIL modules the UWSG wanted to provide the instructions necessary to complete a task and lay the groundwork needed to understand why the task was important.

Due to the independent nature of the USAIL project, a definitive success measure is difficult to define. Because most of the modules written for USAIL are also part of a Computing Education EdCert course on Unix workstation system administration course taught by the UWSG, the group has been able to use the EdCert course as a testing ground. The EdCert course includes in person lab sessions using the lab exercises included in USAIL. The lab sessions are proctored by the UWSG. The UWSG has also kept statistics on quiz questions. The number of times a question is offered and the number of times it is answered correctly are both recorded in a file. This has given the UWSG the opportunity to evaluate the USAIL modules. Is the information too technical or not technical enough? Do users understand the concepts they are working with? Is the information organized effectively? Are users given enough information to be able to perform the lab exercises?

In the short time that USAIL has been available, the UWSG has received a fair amount of email feedback from outside the IU campus system. These messages included requests for information on the project, suggestions for additional topics, compliments, requests to download the modules, and a request to mirror the entire project. UWSG is also using USAIL as a place to refer system administrators when they have questions.

The USAIL project can be accessed on the Web at <http://www.uwsg.indiana.edu/usail/>.

Acknowledgments

Although a single UWSG member was chosen to actually write this paper, the paper and USAIL are the work of the group as a whole. The UWSG includes: Leigh Grundhoefer, Technical Manager; Andrew Cherry, Unix System Specialist; Joe Uthuppuru, Unix System Specialist; and Raven Tompkins, Unix System Support Specialist. The following former members of UWSG also contributed to the USAIL project: Chris Dent, Josh Walgenbach, and Chris Brierley.

Author Information

Raven Tompkins worked as an Addictions Technician, a prop carpenter for an opera company, and in a greenhouse before making her way in the world of Unix. She attended Indiana University, majoring in Fine Arts and Philosophy. In June of 1995 she joined the Unix Workstation Support Group (UWSG) at Indiana University where she works as a Unix System

Support Specialist. Raven can be reached electronically at rtompkin@indiana.edu.

References

- [1] Computer Based Training Systems. "UNIX Technical Awareness." South San Francisco: CBT Systems USA.
- [2] Fiamingo, Frank G. "Introduction to Unix System Administration." http://sunos-wks.acs.ohio-state.edu/sysadm_course/sysadm.html
- [3] Frisch, AEleen. Essential System Administration. Sebastopol: O'Reilly and Associates, Inc, 1991.
- [4] "Introduction to Unix Systems Administration, IFSM 498B." <http://www.gl.umbc.edu/~jack/ifsm498d/>.
- [5] Jones, David. "How do you teach Systems Administration?" <http://mc.cqu.edu.au/david-jones/papers/papers/1993/sage.html>.
- [6] Jones, David. "Teaching Systems Administration II." <http://mc.cqu.edu.au/david-jones/papers/papers/1995/admin2.html>.
- [7] Nemeth, Evi, et al. UNIX System Administration Handbook. 2nd ed. Englewood Cliffs: Prentice-Hall, Inc, 1995.
- [8] "R870: Unix System Administration-A Survival Course." <http://www.washington.edu:1180/R870/R870.html>.
- [9] System Administration Guild. "SAGE Job Descriptions." <ftp://ftp.sage.usenix.org/pub/sage/jobs-descriptions.html> (September 1995).
- [10] "Systems Administration: Installation and Getting Started." <http://www.uwsg.indiana.edu/linux/mdw/LDP/gs/node155.html>.
- [11] "UNIX System Manager's Manual (SMM)." <http://www.eecs.nwu.edu/bsd/smm/index.html>.
- [12] "85321, Systems Administration." <http://mc.cqu.edu.au/subjects/85321/>.

Institute White Pages as a System Administration Problem

Jon Finke – Rensselaer Polytechnic Institute

ABSTRACT

With the planned departure of our mainframe, we had to find a new way to maintain and generate our Institute Telephone directory. This gave us the opportunity to examine every aspect of the directory generation process, and make changes to improve the accuracy of the data, reduce the clerical workload in Telecommunications and Human Resources and eliminate some duplication of data and effort. Given that we already had an Oracle based system to automatically create and remove Unix userids for all employees as they are hired and leave, it seemed that with some minor enhancements, this same system could also maintain our directory information.

To this end, we added a directory module to our Simon Account management system that extracts directory information from the Human Resources database, adds additional non employee information and generates LaTeX source for the printed phone book, HTML pages for the web and a raw feed for the **ph** server. In addition, using techniques developed for some of our system administration tools, we gave both individual staff members, and their departmental administrators, the ability to make changes and corrections to their own directory information, which would not only appear in the directories, but also be reflected in the official Human Resource database. This has greatly reduced the delays and paperwork involved in changing this information, and allows us to have accurate and up to date directories. As an added bonus, many of our “traditional” systems administration tools can now directly query the directory information and include contact info in the displays automatically.

Introduction

For the past five years, the Rensselaer Computing System (RCS), a collection of 700 workstations and Unix timesharing machines available to all students, faculty and staff, has had the Unix accounts automatically managed with a locally developed package called Simon [2, 4] which is built on top of an Oracle relational database. Simon connects on a daily basis to the Human Resources¹ database, and the Registrar’s database (for student information), and based on changes in these databases, creates or expires RCS (Unix) userids as needed. This is similar to the Moira [7] system developed at MIT as part of Project Athena.

During this time, the Institute phone directory was maintained by staff in Telecommunications (part of the Computer and Information Services Division) using a set of custom, stand alone applications that ran on the, soon to be retired, mainframe. To make matters worse, the programmer who supported these applications had retired, and no one else still employed at RPI had ever even looked at the source code. Despite this, this was our most accurate directory database for staff, as each year Telecommunications sent an “update”

¹When Simon was first written, this connection was a payroll tape, once a month. Since then the HR system has been replaced with an Oracle based system called Banner. This is generally known as FAIMS.

form to each department to allow people to check their entries, and return them to Telecom to update the directory. Many people assumed that this was sufficient to keep their personnel records up to date, and never thought to send address changes to HR as well.

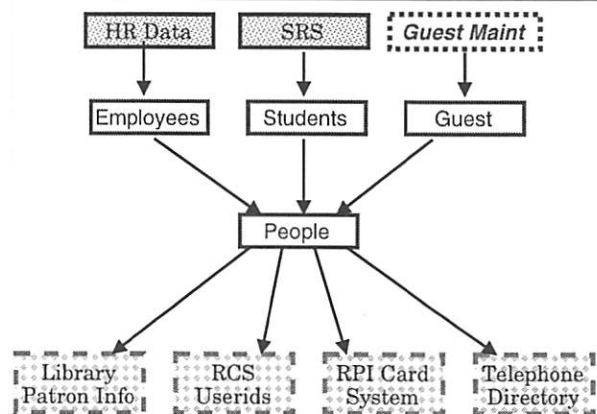


Figure 1: General Information Flow

General Information Flow

For any number of different projects, we have basically the same general data flow required to reach the desired goal. While this particular model applies to the situation at a University, similar models could be applied to Corporate and other organizations.

We have information from Human Resources, which we load into our `Employees` table², information from the Registrar's Student Records System that we load into a `Students` table, and finally, entries into the `Guest` table that are done directly via manual data entry. We actually have a number of types of guests, ranging from Alumni involved in an Alumni computing project, to Adjunct Faculty, contractors, and all sorts of other special cases. Each of these special cases, is maintained by the appropriate staff in other departments as needed. These are cases that do not have a full blown administrative information system to support their operation.

Date from these tables are merged into the `People` table. From this, we can then drive all sorts of other processes, such as the Library automation system patron database, the RCS Userid (Unix Account) management system, the University ID card system (used for identification, food service and access control), as well as the University Telephone directory system. All of these systems need to know when someone has come to Rensselaer, and when they leave, or their status changes in some way.

Directory Objectives

The primary objective of the directory project was to be able to produce an accurate and timely

directory, both in hardcopy and online. In addition, we wanted to establish a single, authoritative and accurate source of directory information of employees and make it available to all administrative units. There were simply too many departments, all doing an inadequate job of maintaining overlapping or even duplicate address lists.

A secondary objective was to be able to delegate control and responsibility for maintenance of different directory elements to the appropriate party, in many cases the individual themselves, or their proxy. While a number of employees regularly used the RCS system, many others did not. However, each department had at least one person who used the finance system (FAIMS), so these people were given the proxy for all people within their department. Although the FAIMS system was NOT part of the RCS system, we had coordinated the userid naming so we were able integrate Simon applications into the FAIMS application menus.

Not all data elements were delegated to the individual level. Some things, such as the official name were controlled by Human Resources, and the actual inclusion of a person in the directory continued to be controlled by Telecom. The data elements are described in Table 1. The "Source" of a data element refers to the source of the data for regular employees. Since we include people, such as emeriti and contractors, in the directory who are not in the HR system, elements can also be entered by Telecom staff. In addition, when the individual is allowed to directly update their own information, their department administrators are also able to make those same updates on their behalf. The Telecommunications department can

²In a relational database, the data are stored in tables. Each table has a number of fields defined, generally referred to as columns. The each column has a name and a data type, and there may be additional restrictions on the information stored in a particular column. Each entry in the table is usually called a row.

Element	Source	Maintainer	Notes
Include in Directory	Faims	Automatic	Set based on status info.
Name	Faims	HR	Never changed manually.
Department	Faims	Automatic	See <i>Departmental Adjustments</i> below.
Title	Faims	HR	Also dept admins on a temporary basis.
Preferred First Name	Simon	Person	Copied back to HR.
Campus Address	Faims	Person	
Campus Phone	Faims	Telecom	Recently made available to dept admins.
Home Address	Faims	Person	Can set <i>Non Published</i> flag as well.
Home Phone	Faims	Person	Can set <i>Non Published</i> flag as well.
Email Address	Simon	Person	Their preferred email address.
Home Page	Simon	Person	
Fax Number	Simon	Person	
Family Info here	Simon	Person	People often list their spouse's name
RPI Class	Simon	Person	RPI Class year for Alumni
RCS Userid	Simon	Computer Center	The <i>userid</i> that may get forwarded.
Email Forward	Simon	Person	Intended for dept admins to forward email sent to <i>userid@rpi.edu</i> .

Table 1: Directory Data Elements

also act as a department administrator for anyone in the directory.

Alternatives Not Taken

Since one of the main objectives was to create one authoritative directory database with distributed updates, a distributed database approach, such as the one at Penn State [1] where the information was distributed among departmental servers, would not suit our needs. There may still be some use for this type of service for some of the lighter weight information such as machine load, but for basic directory information, there would be a single RPI wide source, Simon.

When we established RCS and created a Unix userid for every student and employee, we also established a mail hub with the host name RPI.EDU. People who did not read their mail directly on RCS, could access the mail hub via POP, or simply have their mail forwarded to their preferred mail system. With this already in place, an X500 solution, such as the one at University of Michigan [5] did not seem worth the overhead, for essentially just a query server. The updates were already being handled via Simon³.

Information Flow

One of the early design questions that came up, was if it was possible to generate the directory entirely from information stored in the HR system, or if we had to shadow the data on the Simon system. The eventual decision was to shadow the data to Simon. There were two major reasons for this, the first is that we had to include some non employees in the phone directory, such as ROTC staff, vendors, and others who act as employees but are **not** employed by Rensselaer, and so, not in the HR system. The second reason was that we had to play some games with departmental affiliation (discussion below), and the HR system would not allow for that. Additionally, the HR system is a commercial product, and the folks who maintain that system would prefer to keep changes to a minimum.

Employee Status and Departmental Adjustments

The first problem to solve in producing a directory, is to decide which people are to be included in the directory, and when to take people of the directory again. This is basically the same problem we have solved for RCS userids, ID cards, and the Library system. In Figure 1, we gave a very simplified information flow diagram. In Figure 2, we go into more detail on exactly how we handle the data. Rather than put directory status information into the *People* table, we added a new table, *Dir_Master* to hold the employee status, department, etc.

³When we first started Simon development, we considered an X500 approach to allowing people to update their account information, but instead followed the Moira model of replacing Unix programs such as *chfn* with programs that updated the database directly.

The *Pebempl* table in Faims contains the employee status and department of each employee. This was originally used by the *employees* program to update the *employees* table, and based on the particular status and department, this would be passed to the *people* table and so on to create or expire RCS userids. The *Dir_Master* program using almost identical logic, maintains the *Dir_Master* table which controls if and where someone is to be included in the directory.

The HR (Payroll) system gives us three keys we use to decide if a person will be included by default, their department, their classification (Faculty, Administrative, Exempt, Non Exempt, Student), and finally, their current status (Active, On Leave, Terminated, etc.) For each new or changing employee (from *Pebempl*, see Figure 2) the *Dir_Master* program gets a default include flag value from the *Dir_Departments*, *Dir_Class_Def* and *Dir_Status_Def* tables for each of the keys named above, and if they are all positive, that person is marked to be included in the phone directory. In this way, people are automatically added and removed from the phone directory based on the ongoing activities of the Human Resources department.

Unfortunately, this is not enough. Despite our best efforts, we have to make exceptions, so for each key in the *Dir_Master* table there is a manual override flag. In most cases, this is left null, but if it has a value ("Y" or "N"), that value is used instead of the default value for that key. In this way, we can manually include someone who normally be excluded, or exclude someone who would have been included. There is also a general manual override flag for the entire record, where we can ignore all keys and include or exclude someone. We also include a *review_date* field, that we can set to remind us when to go back and review the manual settings. The big problem with manual settings, is that they are manual and will not change unless someone goes back in and clears them. We need to incorporate a similar field into our Userid management module, as some manual entries have lasted long past their appropriate lifetime.

Once we determined **who** was to be included in the directory, we then needed to determine **where** to put them. A problem with using payroll data, is that it is based on the financial system. While the financial system uses the institute departmental hierarchy, it is driven by the financial accounting requirements. One of the side effects of this, is that most department heads and senior administrators are not paid out of their own department, instead they get paid out of their supervisor's department. This has the effect of moving management out of their own departments. To correct this, we added a *Dir_Master.Dir_Orgn*⁴ field to manually set which department we actually want to

⁴The financial system refers to departments as organizations, generally abbreviated as "Orgn".

list a person in. With this, we are able to put all the Deans and Directors into the right place.

A second problem we encountered with using raw information from the Finance system, is that the finance people have added “roll up” departments to simplify generating financial reports. These extra departments did not have staff and were not part of the “real” hierarchy, and did not appear anywhere outside of the accounting system. This generally resulted in all the staff of a department, appearing in an auxiliary department one level “below” where they were supposed to appear. To fix this, we added a field to the *Dir_Departments* table with the correct organization code for anyone entering that particular department. In this way, any new entries would automatically have the *Dir_Orgn* field set. This also proved useful when departments re-organized and the actual payroll changes lagged (usually to match up with the fiscal year.)

Propagation and Queue Tables

Once we have determined out who goes into the directory, and what department they go into, we need to get the address information, and provide ways to update it. The data falls into two rough categories, the data that originates (at least for some people) in the HR system, such as phone numbers and addresses (see Table 1), and the data that stays in Simon, such as Email address, Home Page, and things that HR didn't

want (or more accurately, did not have a place to store in their system.)

In general, the Simon only information is stored in a table called *User_Dir_Info*. We created a view⁵ of the *User_Dir_Info* table called *My_User_Dir_Info* that just has the data for the current user. This is the same technique we have been using to allow users to set their GECOS field (using a modified *chfn* program), and to forward email sent to *userid@rpi.edu*, using a program called *forward*. Using the *personal_info* program, people can display and update their own (and just their own) information such as their preferred email address, their home page listing, and similar information via the *My_User_Dir_Info* view. This program does do some very basic validity checking on some of the fields, we even go so far to lookup the hostnames in URLs, but other than that, we don't really “care” what people list here.

The processing for information shared with HR is more complex. These changes are queued for later

⁵A database view, looks like a table, but actually depends on underlying tables for the actual data. This data can be from the combination of two or more tables, or of just some selected columns of the base tables. In addition, it can put additional requirements on which rows of the table can be accessed. Views are used extensively in this project to delegate control and access to data.

General Data Flow (Revised)

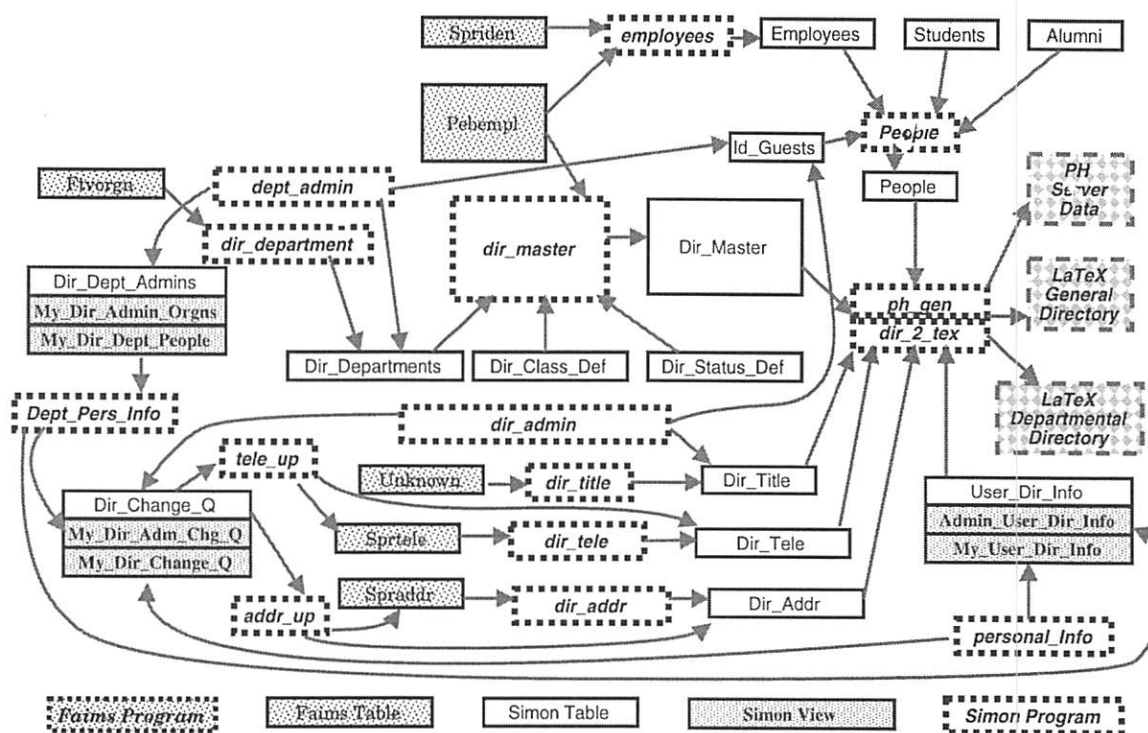


Figure 2: Directory Information Flow

processing in a queue table *Dir_Change_Queue*. The information flow is essentially the same for telephone, address, title and preferred first name information, although we do not actually pass title information back to HR. Both telephone and address information actually break down into either Permanent (Home) and Campus addresses, although this is actually managed by the *personal_info* program and a type field in the records, and the update flow is the same.

Let's look at an address change (but other change types are similar). As you can see in the lower part of Figure 2, if I want to change my address, I use the *personal_info* program, which connects to Simon and puts a change request into the *Dir_Change_Queue* via a view called *My_Dir_Change_Q*. Like the *My_User_Dir_Info*, this only accepts changes for the current user. This change is then applied to the *Spraddr* table in the HR system by the *addr_up* program. Immediately following this, that change in *Spraddr* on Faims, is propagated back to Simon via the *dir_addr* program into the *dir_addr* table.⁶ In this way, the change is effected in both databases. If there is some reason why this change can not go to the HR system (the person may have been terminated, or not even listed in the HR system), the *addr_up* program will skip Faims and update the Simon table, *dir_addr*, directly. We currently process queued changes on a daily basis.

Departmental Administrators

The next part of the puzzle, was to give the departmental administrators the ability to make changes on behalf of members of their department. While many staff may not be regular RCS users, or even have ready access to RCS, each department has at least one person who can access the Finance system (FAIMS). For the purposes of connecting to Simon, the FAIMS system is equivalent to RCS. This also removed the need to provide assistance in connecting to these people; they already had that ability due to their use of FAIMS.

The first step was to create a table of these administrators, and the organizations that they were administering (See *Dir_Dept_Admins* in Figure 2). From here, we created the view *My_Dir_Admin_Orgns*, which is the list of organizations that they can administer, and with this and the *Dept_Master* table, a view of all the people they can administer, *My_Dir_Dept_People*. Finally, we used this to create a view of the *Dir_Change_Queue* table called *My_Dir_Admin_Chg_Q*. With a slightly modified version of the *personal_info* program called *Dept_Pers_Info*, departmental administrators are now able to queue address and telephone number changes for people in their departments. In the same

way, we created the view *Admin_User_Dir_Info* to enable the departmental administrators to update the *User_Dir_Info* table for their staff.

In the actual implementation, *personal_info* and *Dept_Pers_Info* are the same program that use a run time switch to select which set of views to use. This switch also enabled an additional command to allow the department administrator to select which person (from their list of staff), they want to edit. In the same way, the *dir_admin* program is the same program with a different run time switch, that operates directly on the base tables instead of the views. This lets telecom staff change anyone's directory information.

One handy bit of fallout from this, is we were able to add the *forward* command to this program, so department administrators can now establish and changing forwarding of mail sent to *userid@rpi.edu* for their own people. Previously, this would have required the individual to sign on to RCS themselves, or send a request to the PostMaster for this change. While not in the initial plan for the program, it just sort of fell into place for free.

Departmental Information and Virtual People

We had spent a lot of time developing a system that allows us to maintain directory information for people, and to delegate control of that information to other people. We also needed the ability to handle directory information for departments and other organizations. To do this, we simply create a *Virtual Person* for each department. The *person* was inserted into *Dir_Master* with the appropriate Orgn information, so that control of this *person's* information is automatically delegated to the department administrators (just like any other staff member), and they can update their department's address, home page, etc. This required almost no additional development, since routines already existed to edit and extract all of these directory elements. Since these entries are not actual employees, they are not sent back to the HR system, and simply remain in the Simon system.

Along with the departments we obtained from the Finance system, we also manually created departments. In some cases, this was to allow us to insert "external" departments into our existing organizational tree. One example of this is our food service contractor, Marriott. There are a number of Marriott staff working on campus, so we simply created a new organization for them, inserted it into our tree, and assigned (manually of course), the Marriott staff to that department. Since the Marriott employees are not RPI employees, they were entered into Simon as "Guests", as seen in Figure 1.

Generating LaTeX and HTML

The final part of the information flow, is of course the actual directory. We actually wanted a couple of different breakdowns of the data, such as a general staff directory (all staff, sorted alphabetically), the

⁶A convention followed in much of the Simon project is that an Oracle *table* is maintained by a program of the same name.

department hierarchy, and finally a departmental directory with the staff of each department listed with their department. A couple of simple programs were written to extract the information from the database, and generate LaTeX source code for each part. We also found it convenient to generate HTML code at the same time.

Both LaTeX and HTML have a number of special characters that must be avoided or escaped, in order to have the proper things happen when processed. While handling the general boilerplate is easy, some of text that needs to be generated was directly entered by users. Fortunately, we had faced this problem in a previous project that generated HTML from database descriptions [3] and we were able to re-use the `HTMLize_string` and `LaTeXize_string` functions from this program. These automatically handle whatever quoting is needed.

What we wanted vs what we got

Despite months of planning, meetings with Human Resources, and project specifications that were approved by the President's Council, things came up that we did not anticipate, some of which could not be resolved by the publishing deadline.

Conversion Experience

In the past, we had experienced a bit of "title inflation" in the directory. People would send us their titles, and they would be included in the directory. While this might not be their official title as listed with HR, there wasn't any systematic cross checking.

In converting to the new system, we had hoped to switch over to using HR titles exclusively, but we quickly discovered that there was a lot of title drift, and given the time constraints involved, it was not reasonable to get everyone's title changes approved and corrected via HR. We now allow for a manually set title (by the department administrator), and if that is not supplied, we go with the HR title. HR now wants to review the situation, and they are now examining all of the differences in titles between what they have, and what is published in the directory.

Another problem we ran into, was with people who had multiple titles and were in multiple departments. In the first release of the directory, we did not allow this, as HR was not supplying that information. In some cases, people had their titles set manually to *title1/title2* within the limits of the size of the title field. We are investigating ways of allowing multiple titles and departments, but since they will be manual entries, we do not plan on encouraging their use.

Data Cleanup

Part of the conversion process involved cleaning up the directory data. We merged some data from the old directory, but frequently found that the old directory info did not match the HR data for that person. Then we had to go back to the individuals and ask

them to verify their information. In the past, the Telecom folks had sent out departmental listings to the departmental administrators and ask for corrections. To this end, we modified the departmental directory generation program to generate a departmental entry and all the staff entries for that department, and a source file that would run that through LaTeX, as well as generate a set of mailing labels. This gave us a nice cover letter to the department administrators explaining how THEY could update the department and staff information via the `Dept_Pers_Info` program, as well as a page per staff member listing each directory field with their current information. This format made it very clear to the individuals what fields were available, since some of fields had not been collected in the past. The department administrators had the option of distributing the individual sheets to their staff, or doing the updates themselves. In any event, these updates were done at the individual or department level, and NOT by Telecom staff.

The other part of the data cleanup involved the campus address. Historically, the campus address was generally three lines, with the department name on line 1, the room number and building name on line 2 and the text "RPI" on line 3. Unfortunately, there was no consistency in the entries, making it impossible to generate either building or department directories. (There were about 1200 unique department names when we started, we are now down to 177 departments.) To clean this up, when someone enters a campus address change, they have to select a building from a list of buildings, and then provide a room number. The department is automatically obtained from the information in `Dir_Master` and `Dir_Departments` tables. In addition, we also include an internal database building id, that gets propagated to the HR system (they agreed to modified their tables for this one), and back to Simon again. After giving the staff a chance to correct their addresses, we added an option to `Dept_Pers_Info` that displays each "unclean" address, and prompts for a new address. This allowed us to clean up the remainder of the addresses in a very short amount of time.

Intended Results

Despite these problems and some last minute changes, we ended up with a 148 page telephone directory with both alphabetical and departmental listings, as well as a solid directory feed to our **ph** server⁷. In addition to the **ph** database update, we also do a nightly refresh of Departmental Directory web pages <http://www.rpi.edu/AutoGen/deptsum.html> and

⁷Our **ph** server is a package developed at the University of Illinois at Urbana, and has not only a Unix server, but clients for many platforms. It was originally developed by Steve Dörner (sdorner@qualcomm.com), Qualcomm, Inc and is now maintained by Paul Pomes (p-pomes@uiuc.edu), University of Illinois Computing and Communications Services Office.

the list of Institute phones (an alphabetical listing of departments and services) <http://www.rpi.edu/Auto-Gen/univtel.html>. Both of these pages include phone numbers and fax numbers (when available), and an link to the department's web page, if it exists. The department directory also includes a link to a staff list for that department.

In doing the Institute Phones directory, we also needed to include services (for example, Computer Repairs). These are added in to another table, and where possible, we indicate which department "owns" that service. In this way, we can include those services at the top of the staff listing for that department, and also as a sub entry in the Institute Phones Directory. Where we encounter an email address for a person, department or service, that email address is automatically inserted into the web page with a `mailto: URL`.

Another nice thing we gained from the project was the ability to collect and publish "less important" data, such as a person's fax number, their home page, etc. Before the `personal_info` program, it was simply too expensive in terms of staff resources to collect and enter this information into the directory. Now that individuals can provide this themselves, we can include more types of information in the directory. In the case of home pages, we don't include them in the printed directory, on the assumption that anyone looking for URLs will be looking online where we do include them.

Unexpected Benefits

This project had a number of side effects that we really hadn't considered in the initial design, or at least were not of major concern at the time.

One of the first departments to benefit from the new directory system was the parking office. Since we now had all employees listed with their campus address in a consistent format, we were able to generate lists of employees, by building. Rensselaer has a number of parking lots located in different areas of the campus, and staff generally prefer to park in the one closest to their workplace. To figure the parking lot assignments, the parking office used to manually sort these lists of employees. With clean address information in the HR database, this three day project turned into a simple database report.

Another area that improved, was our general system administration management tools. We have been managing the host database, software service contracts, and RCS userids with the Simon database. These tools are now able to automatically include the address and phone numbers of the people involved. This has been very well received in our network operations center, where they can now just enter a single command to look up a local host, and get back not only all the host information, and the name of the owner and system administrator, but their address and phone number as well.

The phone directory also includes a building and departmental abbreviations. These abbreviations are stored in the *Buildings* and *Dir_Departments*, and are automatically used when generating directory pages (or hardcopy), and we also extract them as lists to include in the directory itself. This makes it very easy to change the name or abbreviations used for a building or department and ensures that all parts of the directory use a consistent set of abbreviations.

Another application was driven by a new state law that requires RPI to report the names and address of all employees. While this wasn't a problem for the regular staff, Rensselaer hires a large number of students. HR never bothered with addresses for student employees, as all paperwork was handled in person. Given that we had a path in place to update addresses, it was simple to write a program to extract student address information from the Registrar's files (since we already were loading student data into Simon), and send "address changes" into the HR system.

We continue to find administrative applications that can now benefit from accurate and retrievable address data. One recent example of this is the letter distributed to new students telling them who their faculty advisor will be. We can now include their advisor's campus address, phone number and email address. Previously, this was not practical.

Future Work

There are still some parts of the directory that have yet to be fully integrated into this system, such as the Emeriti/Retiree address list, the Student Clubs and Organizations listing, and so on. It turns out though, that this directory system gives us almost everything we need to support these additional listings. In general, we simply create new organizations, which gives us the ability to delegate control over those organizations to others. We are defining one new construct though, an *Aux_Member* table, so we can assign any arbitrary person in the database as an auxiliary member of an organization. We can include these people when generating department listings, and this will likely be used to handle the problem of people who need to be in more than one department.

While we can delegate the ability to add auxiliary members to department administrators, since we are not using the *Dir_Master* table, a department administrator could not add someone, and then update their address information. This allows us to maintain better control over who can make address changes, yet still allow us to delegate auxiliary membership for specific groups. This will be especially useful for the student clubs and organizations, since student address changes are handled via the Registrar's office (using an earlier and different online system). It will be possible for club officers to maintain the membership lists of their clubs, and then we can generate membership lists that include addresses and status information drawn directly from the Registrar's files.

The really big change to the directory system however, will be migrating the `personal_info` to some sort of WWW interface. General computing at Rensselaer is moving away from Unix workstations, and it appears that the Web will be the best common denominator for user interfaces. We are also faced with changes to the Faims system that will require that the `Dept_Pers_Info` program be converted to a real Oracle Form (so it can work in a Client/Server environment), or be removed. We have a number of other personal information services that are available via RCS that we will also want to make available via a web interface, so we will be exploring how to authenticate web sessions connect to the database during the upcoming year. Fortunately, there seems to be a lot of activity in that area, so hopefully we will not have to roll our own solution.

In an effort to make directory information available to more platforms, we are investigating putting up an "X500 Lite" server [6], as described developed by the Center for Information Technology Integration at the University of Michigan.

References and Availability

All source code for the Simon system is available for anonymous FTP. See <ftp://ftp.rpi.edu/pub/its-release/simon/README.simon> for details. In addition, all of the Oracle table definitions are available at <http://www.rpi.edu/campus/rpi/simon/misc/Tables/simon.Index.html>. Directory pages can be seen at <http://www.rpi.edu/AutoGen/deptsum.html> and <http://www.rpi.edu/AutoGen/univtel.html>.

Author Information

Jon Finke graduated from Rensselaer in 1983, where he had provided microcomputer support and communications programming, with a BS-ECSE. He continued as a full time staff member in the computer center. From PC communications, he moved into mainframe communications and networking, and then on to Unix support, including a stint in the Nysenet Network Information Center. A charter member of the Workstation Support Group he took over printing development and support and later inherited the Simon project, which has been his primary focus for the past five years. He is currently a Senior Systems Programmer in the Server Support Services department at Rensselaer, where he continues integrating Simon with the rest of the Institute information systems, and also deals with information security concerns. Reach him via USMail at RPI; VCC 319; 110 8th St; Troy, NY 12180-3590. Reach him via electronic mail at finkej@rpi.edu. Find out more via <http://www.rpi.edu/~finkej>.

References

- [1] C. Mic Bowman and Chanda Dharap. The enterprise distributed white-pages service. In *USENIX*

- Technical Conference Proceedings*, pages 349-359. Penn. State University, USENIX, January 1993.
- [2] Jon Finke. Automated userid management. In *Proceedings of Community Workshop '92*, Troy, NY, June 1992. Rensselaer Polytechnic Institute. Paper 3-5.
- [3] Jon Finke. `Sql_2_html`: Automatic generation of html database schemas. In *Ninth Systems Administration Conference (LISA '95)*, pages 133-138. Rensselaer Polytechnic Institute, USENIX, September 1995. Monterey, CA.
- [4] Jon Finke. Relational database + automated sysadmin = simon. Boston, MA, July 93. Sun Users Group. Invited Talk for SUG-East 93.
- [5] Timothy Howes. Integrating x.500 directory service into a large campus computing environment. In *LISA IV Conference Proceedings*, pages 125-132. University of Michigan, USENIX, October 1990.
- [6] Timothy A. Howe. The lightweight directory access protocol: X.500 lite. Technical Report CITI TR 95-8, University of Michigan, July 1995.
- [7] Mark A. Rosenstein, Daniel E. Geer, Jr., and Peter J. Levine. The athena service management system. In *USENIX Conference Proceedings*, pages 203-211. MIT Project Athena, USENIX, Winter 1988.

New Fangled Phone Systems Pose New Challenges for System Administrators

Snoopy – iXOS Software GmbH

ABSTRACT

Recently I have noticed a remarkable shift in the way companies deal with telephone systems. More often than not the selection, installation, programming and above all administration are put in the hands of the systems administration groups. This paper gives an introduction to the architecture and features of NFPS (new fangled phone systems) which are rather different to the usual systems we have to take care of. Additionally some of the new language you will encounter is explained. And to round it off I will try to relate some lessons learned on installation day as well as how we utilize features of the new phone system to improve our daily system administration.

Background Information and Problem Definition

iXOS Software is a smallish German software house. We currently support a user population of about 200 users in our headquarter in Munich with a sysadmin group of five. We run a large network with some 500 machines, PCs and workstations as well as numerous WAN connections in order to give our users Internet access and connectivity to our subsidiary companies in Leipzig (Germany), Switzerland, Prague (Czech Republic) and the USA (Belmont, CA, Dallas and Philadelphia).

Our old telephone system (PBX) was made by Siemens-Nixdorf, a large German manufacturer of such systems. The PBX was entirely analogue, although we could and did connect some ISDN lines in order to give our directors phones with more flexibility. We were faced with the rapid growth of our company and the old system was simply just getting full: all the available expansion slots for line interfaces were getting full.

We were not happy with the quality of the voice transmission: customers complained the quality was lousy, too low in volume, hissy voice transmission etc. Dialling became a painful exercise: as the cabinet filled up, it took the processor of the PBX longer and longer to poll all the boards and so it took several attempts at dialling a number for the PBX to notice it was being asked to dial.

Additionally we were annoyed at the financial problems arising from running the old phone system. The price of the expansion cabinet with a suitable processor upgrade ran to well over 100k \$. Over the (five year) lifetime of the PBX the price for an expansion board (offering a mere five phone lines) rose from about 1,000 \$ to over 2,500 \$. We felt this was the wrong direction the prices were moving given the prevailing technological trends and this certainly did not increase our customer satisfaction.

So we had to either bite the bullet and invest heavily or invest heavily and bite the bullet and decide to go for an entirely new phone system. As market deregulation has begun in Germany we were confident we could find a manufacturer of PBXes who could deliver a suitable and cheaper solution.

Basic Requirements

Our new phone system was meant to provide the following features:

- Fully digital transmission of voice in house and to the outside world (via ISDN)
- New, sexy phones with a display capability
- Good integration into our network environment (i.e., we wanted the PBX to speak TCP/IP on an Ethernet)
- Geared for CTI (Computer Telephony Integration)
- Easy administration – we want to be able to do most of that ourselves
- Advanced features – like hotlines, call vectoring etc.
- Voice Mail system
- Distributed systems – we wanted the architecture to deal with the rapidly expanding company – over several buildings on different streets and sites which can be several hundred miles from each other.

So we went out and contacted some manufacturers asking for an offer. The initial selection of manufacturers was done by simply asking around: we deal with a large number of suppliers who do a lot of selling by phone and I asked them what make phone system they were using and also if they were happy with it. This gave us an initial list of seven relevant manufacturers – these were invited to present their wares and introduce their systems.

Buying a phone system is very similar to having a baby: you are suddenly confronted with the workings of a whole new industry with its own rules and

vocabulary. The experience is very similar to buying a pram: you walk into the shop with your spouse and you found a pram of suitable size, shape and styling and price tag (\$499, special deal). Bliss. Then the attentive sales lady asks you: don't you think you will need an umbrella (to shield baby from the sun), yes of course. And a collapsible roof against the rain (yes, that too). And a fur sack to keep baby warm in those awful European winters. Sigh. Yes, by all means (which in this context means – your means). By the time you have completed the system configuration and scaling you have exceeded the initial budget by a factor of about 2.5 and have also purchased mission critical sub-systems you never knew existed, let alone deemed necessary.

As said: enter the realm of phone systems as a proven but mainly UNIX oriented system administrator you find yourself in a terribly similar situation – an irrevocable process (your boss) takes over your life by delivering a new, utterly unknown system (the NFPS) onto your doorstep and you have to live with it for a long time and give it your best shot.

So lets go through the features we wanted in more detail and explain some of the language along the way.

To ISDN or not ISDN?

Firstly the connection to the outside world: well we went for an ISDN line known as a PMX – primary multiplexer. I believe in the US this is known as a T1 link. Basically it provides 30 ISDN channels of 64 kbps to an aggregate bandwidth of 2 Mbps. In Germany each 64 kbps channel is known as S0 (S zero) and the T1 is known as an S2M (affectionately also simply referred to as a PMX).

We were considering for a while to equip each and every desk with a proper ISDN phone. This would have given us some nice features across the ISDN network (like for example automatic call back if the person we want to speak to is currently busy). However it would have meant rewiring the buildings with four wire phone sockets because an S0 ISDN socket needs four wires. Some manufacturers allow you to run a two wire phone and they then mimic ISDN over these wires. However the needed adapters drive up the price of a phone by about 15-20%.

Furthermore we were thinking of equipping the PC workplace of our users with ISDN cards and using the ISDN line to the desk both for telephony via the PC and data transmission. Since such PC ISDN cards are notoriously fussy when it comes to the ISDN parameters we were not confident that foreign manufacturers of the ISDN mimicry gadgets would be accurate enough to satisfy the cards.

ISDN phones turned out to be expensive beasts and the interface cards needed in the PBX were more expensive too (higher price, less lines per card). Added with the re-wiring costs (estimated to be about

\$70,000 for our site) plus the risk of not being a good solution for the PCs anyway we decided to abandon the ISDN to the desk idea fairly early on in the game.

Basically this coincides with the manufacturers interests: they are true traitors of the ISDN concept because making ISDN on the desk cheap would create an open system where you could cut out the phone system and leave the phones out there. This would give you a lot more flexibility to throw out a phone system you don't like. But the manufacturers make this a ridiculously expensive option and point out with coy batings of their corporate eyelids that their two-wire digital phones are vastly cheaper than the ISDN solution. But of course you are then stuck with their phones too and are locked into their proprietary digital protocols: you cannot then change manufacturer of the PBX without changing all the phones. This makes a change prohibitively expensive.

So the pressure is on: once you have made your choice of phone system you can be assured that it will be around for a lot longer than your run of the mill UNIX workstation which gets thrown out after a year or perhaps two. It is therefore vital that you buy yourself into a corporate strategy which will fit your plans for years to come. Take into account the growth of the company (iXOS grew at 50% p.a. since its founding in 1988) and you have to find a PBX which will scale well and handle the anticipated loads. Serviceability, availability of spare parts, training and administration all become very much more important. The investment into the PBX is not only significant due to the amount of money changing hands but also due to the time involved before you can ever change it: therefore it is wise to invest a lot more in the training of the admin staff etc etc. for such a system.

Incidentally: as not all public switches are digital yet in Germany, we have kept four analogue lines as a backup – and indeed our local exchange is analogue which means within our area code we can only be reached via the analogue lines from the immediate vicinity.

Basic Features of PBXes

PBXes are usually UNIX based systems – usually PECOS UNIX which is a real time UNIX. variant. Before you go overboard with the naive view that PBXes are just like any other UNIX system you can handle you have to be aware of the fact that the UNIX system is almost entirely hidden from your grubby palms. All manufacturers supply a custom built interface – screen/mask oriented. A shell is not available (and none of the manufacturers I asked knew what a shell was...)

In order to administer such a system you have to get certain passwords from the manufacturer: these are part of strictly controlled hierarchy. The passwords which are akin to the root password (i.e., you can do anything) are not available.

In most cases the customers are only allowed very little access and the service department of the manufacturer handles most change requests via remote modem access to the PBX. However most of the manufacturers I spoke to came in one of two categories: those who said its their phone system and they let you use it and they do all the admin. This is not such an enticing option as it might sound for their response time may be vastly different to what you want and your users expect. So the second category of manufacturers says: we spent a lot of time and money giving you an interface for you to handle the PBX and we expect you to do most of the administration yourself. This is a good idea (the one we preferred) for it means our level of service is also good for the phone system (which may or may not be better to what the manufacturer can offer...)

Hardware Features

On the hardware side of things the PBXes are essentially one huge backplane of some industry-standard bus system (usually VME). Boards can be inserted and taken out while the systems are running: special backplane logic and power wiring makes sure the boards do not get fried or the backplane hangs during the process of insertion or removal. The bus bandwidth is quite high because you have to shift data packets quickly across the back plane in order to properly transmit voice. Packets have to be delivered in sequence and connections have to be made quickly. Hence most boards have own processors (typically M68000's which are now cheap commodity chips).

Special boards provide an extension of the system bus over fiber-optic links to more remote expansion cabinets (known as carriers). This means the PBX can be viewed as a master carrier with sub-carriers which have no local intelligence bar the on-board processors of the interface boards. Such an architecture is very flexible: picture the scenario where the Telecom people tell you something like: we are terribly sorry, but we have no more line capacity for the feed of your building with the master carrier for the second set of ISDN lines. We could only give you lines to the building on the other end of your campus.

This is no problem as long as said building has a sub-carrier: you simply stick the ISDN interface board into the sub-carrier and the main carrier will route calls over the optical fiber to go out on the other ISDN lines provided the ones attached to the main carrier are all occupied.

PBXes are designed for 100% uptime: hence you can stick boards in while the system is in use: the PBX auto-detects the new board and away you go. Of course a failure of the CPU board or other critical components will stop your PBX dead in its tracks but this only happens rarely. Its advised to hook the PBX up to an uninterruptible power supply (UPS). This increases the lifetime of the system and also allows

phone calls to continue for a few minutes if the power fails (the phones are powered by the PBX through the two wires).

More Terminology

Some more terminology to keep you happy: interface boards have ports. These are the two digital wires which lead to the terminal which is the fancy way of referring to a phone. Ports are usually but not necessarily associated with extensions – an extension is the set of digits you dial to talk to a principal which is the person sitting there waiting for the phone to ring. Most PBXes will support virtual extensions – this is an extension without a port. This is actually a lot more useful than it sounds: they can be used for employees who now work off site but still want their extension to be valid and to receive voice mail. But you can free up the port and give someone else a real phone.

Sometimes a phone/principal combination is referred to as an agent – sometimes agent is considered to be a synonym for a principal – this is often a context sensitive distinction. Often agent is referred to a somewhat lesser breed of phone user than a principal – often implying something akin to a boss/secretary relationship. Agents come in droves known as splits or hunt-groups which are extensions grouped logically together by the PBX to do fancy things (explained in more detail later).

Creeping Featurism

Modern PBXes are packed with features most of which you will hardly ever need. In fact most manufacturers nowadays refer to their systems as Call Centers. The difference between the two is essentially marketing driven.

Basic features of all examined call centers are things like call forwarding – you can just route your phone to another extension, i.e., where you will be seated for the next two hours of Solaris install.

Auto-Call Back means if an extension you call is busy, then you activate the ACB button. As soon as the desired extension puts the phone down, the PBX will ring your number and his/hers and connect you. The display reflects this saying something like "Call back to Snoopy".

Our stations have three so-called call appearance buttons (with associated LEDs). When a call arrives then the display gets activated and the call appearance button lights up. The third call-appearance button is labelled a restricted call-appearance. This means normal calls cannot arrive there. Consider if you have two calls on your extension and you want to transfer one or both out of your station to someone else. In order to do this you have to get a third free line (this is somewhat analogous to the last free process slot in UNIX being reserved for the super-user) to call the other station and transfer the call. Hence you do not

want a normal call to arrive on the restricted appearance lest you will not be able to get rid of any calls short of putting the darn phone down.

If you are talking on one line then your station is considered to be active – if you have all lines (you may of course have as many call appearances as you wish or have buttons for) handling a call except the restricted appearance only then are you considered busy which means the PBX will not hand you an additionally arriving call but will return a busy signal to the caller.

Some stations (like us system admins, he, he, he) can place priority calls which means we can wind up on the restricted call appearance and/or punch through any call forwarding or other cloaking devices the principal has enabled. This is really useful if you have to notify a user that you have to take his/her workstation down immediately because its misbehaving on the net or whatever.

Conference Calls are also a nice feature – you can get up to six different people on the phone all at the same time. Our directors make frequent use of this to have phone meetings amongst all the directors of our subsidiaries. In Germany incidentally it is forbidden to set up conference calls off-switch which means we are not allowed effectively to call external numbers and get them in the conference. This law is an ancient relic and a source of great amusement even to the Telecom employees who cheerfully admit its an idiotic law which currently cannot be policed anyway because a conference call is not different to their switches than a normal call. Of course this law will go away with increasing deregulation in other words by 1998.

Every extension has associated with it a coverage path – this is a path a call can take if the principal does not pick up the phone. Our default here is to route calls directly into the voice mail after a number of rings (typically five rings.) Of course you can mention other extensions in the coverage path, up to three such points are allowed in a coverage path and you can also chain coverage paths together. Alas while you can decide when you want a coverage path to take hold for all calls, external calls only or when you are active or busy or whatever there is no way to have two coverage paths associated with an extension: one for internal calls and one for external calls.

A typical coverage path for sales people goes to the sales assistant (if the primary target, i.e., the extension that was dialled) does not pick up. Then if the sales assistant does not pick up the next point in the coverage path is the switchboard. Only if the switchboard is unable to pick up does the caller wind up in the primary targets voice mailbox. Germans are not well accustomed to answering machines and we find that its way better to humanly handle calls because otherwise customers just put the phone down.

Our phones used to have a button we programmed to be the send-all-to-cover button: all calls would directly drop into the coverage path (which for most of our developers meant voice mail). Some of our developers are real hermits and had the send-all enabled all the time, self stylized paranoiacs. So soon our customers started complaining, that our folks were not reachable (and some of our developers will not read e-mail for weeks or also ignore little red voice-mail lights on their phones for weeks on end). So we took their send-all buttons away from them to enforce managements idea of reachability by phone. This created a bit of a upheaval but the waves of discontent soon ebbed.

Of the more razzly-dazzly features voice mail is actually terribly useful and the voice mail system provides a good interface to actually forward voice mails and file them away in complex directory structures. Opposed to normal e-mail, voice mail has an automatic expiry (set at two weeks at our site). Of course voice mail is yet another biblical plague – you now have an additional mailbox to wade through.

Hunt groups are useful too: you can designate bunches of extensions to be considered to be linked together. Consider two extensions 1122 and 1150 to be in a hunt group. Someone calls 1122 but nobody there picks up the phone. The PBX will then try to ring extension 1150: this implicit routing is transparent to the user. You can also designate a bunch of extensions as a terminating extension: when a call comes in for any one extension of such a group all of these phones ring at the same time! This is actually fairly spooky to have about 15 phones go off at once in a large office. It creates instant panic and is the poor mans implementation of Captain Kirks red alert.

Hotlines and ACD Functions

One really good feature of modern call centers are ACD (Automatic Call Distribution) functions to support hotlines or telesales applications.

ACD distributes incoming calls evenly and fairly over a group of extensions organized in a split or hunt group. Usually agents have to login to the split in order to be recognized that they are available to be allocated calls. Incoming calls are divided fairly amongst all logged in agents. Due to this login process the PBX knows which agents are actually staffed.

An important accessory feature to ACD is after-call – this allows you to specify a time after the completion of a call until the agent is made available to take calls again. Its a sort of delayed login after a call. Imagine a scenario where your sysadmins take calls from the users and have to enter a request into a request system: the PBX takes the agent out of the split for a fixed time after the actual phone call (say three minutes) to allow one to fill out forms or the screen mask or whatever. When the after-call time has

elapsed, the agent is automatically put back into the split in available mode.

Good PBXes provide detailed reports on the throughput of the ACD groups: how many calls are taken an hour, the average wait time, the longest wait time of a call, the number of failed calls (a failed call is one where the caller puts the phone down before an agent can take the call). Such statistics allow you to properly size the number of agents in a split to handle calls. Such things are not very useful for a small fry company like iXOS but its very useful for professional telesales companies for example in case of adverts on TV ("Call now to order your Super Thigh Stepper for the smashing price of only \$139,99!"). Such adverts generate tens of thousands of calls within the two minutes after the advert. So its essential to have a powerful call center in place capable of handling such loads.

ACD functions are best augmented by advanced call vectoring features. Call vectoring enables you to program a path (N.B. don't confuse this with a coverage path) the call is meant to take amongst different splits of agents and to be treated differently. For example, you can program the vector to take into account queue lengths of the split thus enabling easy overflow functions. Some PBXes only allow such overflow functions by going down to the PBX and taking a screw driver and twisting a potentiometer to shift incoming calls from one split to the other if one of them is being flooded with calls. This is far too rudimentary; we really wanted to handle this type of problem in a better way by appropriate programming.

Vectors can take the queue length into account differently too, for example you can test "if calls_waiting in split > 3 then deliver busy". This would cause the fourth caller into a split to get a busy signal.

Other vector primitives include functions like testing if a call is external or internal, checking the time of day etc. This is useful for handling and routing calls appropriately. For example our weekly sysadmin meeting is on Tuesdays between 11:00 and 12:00 a.m. - we have therefore programmed the hotline vector to play a special announcement during this time so that users will fall into the voice mail immediately so we can cope with the problem after meeting.

Vectors can be interactive - you can play an announcement ("Please type 1 to leave a voice mail message, type 2 to speak to someone directly") and then gather the input and branch accordingly. This can also be combined with the announcement of the queue length ("The average wait time is currently 3 minutes. If you wish to stay in the queue please press 1")

Advanced PBXes allow you to integrate a voice recognition system into the vector programming. This is an expensive co-processor running to the cost of about \$50,000 extra. However this can be very useful to speed up customer service in large call centers (airlines tend to be very fond of these things).

Vector programming is a bit of an art and a different etiquette applies. In my first attempts I actually did some very rude things like delivering the busy signal after people hung around on the phone for two minutes, paying the bill and listening to music. Well, it was a bug in my vector admittedly but the users reactions did make it very clear what I should avoid in the future!

Advanced vectoring allows you to also change the priority of a call in the queue depending on the number of the caller (this is only possible on an ISDN network where callers can choose to transmit a packet containing their ISDN number). This feature is called ANI (Automatic Number Identification). In a vector you can test for the number and give it special treatment. For example SAP (a company that holds a 10% stake in iXOS) has the number 06227-34-0. We use this number identification to place their calls to our hotlines in a privileged position because they usually need quick service.

ANI can be used in conjunction with CTI (Computer Telephony Integration). The vector allows you to specify to send a packet out on the Ethernet when a certain number has called. This packet is then received by a server which allows you to place information on the principals PC screen.

Some things we find lacking in the methods of vector programming is testing and setting sort of "environment variables". As an example take a company that wants to introduce a "quiet time" where all calls from the outside should go to the voice mail and not disrupt the quiet workings of the company in top gear. I would like to code a vector in the form of:

```
if ($quiet) and
  (daytime is 14:00 - 16:00) then
  route all calls to hunt-group 10
  /* our AUDIX hunt group */
```

Then in the main systems menu I would just set a variable quiet and things are enabled. If we deem the experiment a failure I just unset the global variable and I never have to go back and change any of the vectors. OK, I admit I am heavily influenced by the UNIX philosophy but I would really enjoy it if this philosophy would not be as well hidden by the administration interface.

Connectivity and CTI

This brings us to the issue of the PBXes connectivity with our LAN. We have an ethernet card installed and some applications running on the PC. Actually the issue of ethernet connectivity was a real problem for most manufacturers: of the seven applicants only two (Ericsson and Lucent Technologies (ex-AT&T)) provided a satisfactory solution. All the remaining suppliers either gave us a disbelieving stare ("Why on earth would you want to have an ethernet connection to your PBX!??") or came up with

horribly kludgy "solutions" – the favorite one was: stick another ISDN card into the PBX and stick an ISDN card into a PC. Then the PBX sends its packets to the PC which in turn plays ISDN to Ethernet bridge. This however implies that the reliability of my Ethernet connection is that of the PC and to us who know the full significance of such know that this as a totally unsatisfactory solution. No thanks.

Connectivity and CTI are based on two standards TSAPI and TAPI. The latter is supported primarily by Novell, the former is the Microsoft standard. Our PBX is adherent to TAPI and we are in the process of evaluating administration software which runs on Solaris 2.X as well as front ends running on Windows PCs.

The front ends for the users offer nice integration of the PBX with the PC. Voice mail and dialling are available through the PC. Additionally you can maintain a phone and address book on the PC and dial with a mouse click.

The voice mail integration is also really nice: a voice mail message appears as an icon on your screen. Moving the mouse over the icon will enlarge it to show the statistics associated with the message: date, length, sending extension or number etc. Double clicking the message icon will deliver the voice mail by letting your phone ring and playing you the message. You can then also manipulate the message (deletion, filing, forwarding etc.) through pop up menus.

We are also planning to use the interface to the PBX to connect it to our in-house databases – here we record our sales leads and customers etc. We want to be able to dial a customers number using a function key in the display. If a customer calls the sales people we want to be able to display his/her name on the phones display and also pop up the correct customer information mask on their screen. However we have not done this yet as we are still developing the software for this; this is work in progress.

In future we would also like to use the TCP/IP connection to the PBX to be able to send faxes directly from any PC or workstation. The reverse channel should also be possible. Our aim is to also hook up all of our modems to the PBX and access them via the ethernet link (i.e., using the PBX to act as a terminal concentrator).

PBX manufacturers want to integrate the phone entirely into the PC: your voice packets get carried to the PBX over the ethernet and then out onto the public network. The aim of the PBX makers is clear: they want to get more money out of the software as their fiscal base for charging premium prices for PBXes is being eroded by more and fiercer competition. So they want to slowly shift their revenue base to include more services and software.

The PBX makers offer other reasons for this being a good idea: you save space on your desk, you get a lot more comfortable access to all kinds of

wonderful functions. For example it is planned to allow every user to receive faxes on their extension: the CTI software will then display the fax bitmap on the screen.

However we are opposed to this idea for several reasons.

We believe a phone should only be augmented by the PC but should stay a separate appliance which is functional as a separate device. We support this concept for different reasons mainly that of reliability and usability.

Reliability is a big problem. PCs crash, PCs break, users fiddle their setup files and their reliability and availability is usually far inferior to properly administered servers and workstations. Furthermore power problems arise: its impossible to hook up all our PCs to an UPS for example. It would be prohibitively expensive. Our current setup allows us to run the phones on an UPS. We believe our sales people and other heavy phone users would be very upset if they would not be able to use their phones if their PC is dead. That simply happens too often for comfort.

Usability is another issue and can have possibly legal repercussions. Phones have been around for over one hundred years and most people in the civilized world have no problem walking up to a phone and using it. However imagine a scenario of your office mate suddenly grovelling to the floor with a heart attack and you have to wait until your NT workstation boots until you can call help.

We believe, even if it sounds awfully conservative that phones will be around for a while yet. CTI offers fantastic possibilities to increase your performance using the phone but it will probably never replace the phone on the desk entirely.

Administration

The administration of modern PBXes is usually done through a channel PBXes know how to handle best: phone lines. You are supplied with two modems: one for you and one for the service department of the manufacturer. This is indeed very handy and we are quite happy to let them adjust critical parameters of our PBX (like tuning timing parameters for the ISDN cards and other things at a very deep technical level).

Administration is done via a terminal emulator over the phone line and you then talk to a screen based administration tool which hides the underlying phone system entirely. Soon we want to use a Solaris server on our network to run a Solaris based admin tool which talks to the PBX over the ethernet. We hope that we can do all the administration of our switch, the AUDIX Voice Mail and additional functions like the call charge metering through the ethernet. Also we need additional scripting functions so we can specify configuration commands like "remove the send-all-

calls button on all stations of type 8410 and replace it with a date-time button.”; when we enforced management policy to take away the send-all-calls button we actually had to do this by hand which is a mindblowingly tedious way to spend a morning. Once you have done that you begin to realize that things like *sed(1)* are very useful tools indeed!

Security

PBXes allow the system administrator to create privileged ports – these are allowed to access privileged functions of the PBX. The idea is that you can do some switch administration from home without a modem just by using a touch tone phone. PBXes allow most functions to be accessed not only through programmable buttons on the phone but also through key sequences referred to as feature access-codes(FAC). For example you can issue a call redirect (call forwarding) directive by typing: *04342433#. This will forward all calls (*04 is the FAC) of extension 342 to extension 433. The # is the end delimiter.

From home you would then call the privileged port (a virtual extension). You then have to validate yourself with a password (a number of three to eight digits length). You then get a line to the PBX and the world is effectively your oyster. Small wonder then that such ports are frequently the target of crackers who specialize in breaking into PBXes. Apparently in the US this is big business to defraud companies this way. The crooks get hold of the number of the privileged port(s) and then rent an apartment and ask the local Baby Bell to put in lots of phones. Then they accost customers in the street to the tune of “Phone as much as you like for only 20 bucks” – they then ring the privileged port of the hacked PBX and get an outside line and the company running the PBX pays the bill. This can go on for several weeks before anyone notices.

For this reason the system allows you to create such ports only with a limited lifetime – you can specify “number of logins” or a lifetime in days. In case of repeated login failures the port is automatically disabled and a message gets logged to the system logs.

Of course other security issues arise: your PBX is on the Internet if your companies LAN has a connection there and you have to ensure that nobody can break into your LAN and use the phone system. People could route Internet packets through your PBX and fax out or just use the phone and you pay the bill. Not a good idea.

On top of that your very own employees can defraud you much easier. They could for example call-forward their phone to their aunt in Australia and then they go home and call their extension from home. You pay the bill to Australia. Not good.

Also users can forward their voice mail off-switch so that voice mails get delivered to your answering machine at home. This can also incur heavy

costs if they do this to another continent. If the voice mail system they forward their voice mail to is also configured to forward the voice mail (perhaps back to the switch it came from) you can create costly mail loops and overflowing voice mail disks on both sides of the Big Mud Puddle.

Anybody who knows your extension number can listen into your voice mail: you can protect it with a password (a number). This is a good idea. We had one incident of a rather delicate nature where a user walked up to someone’s phone to actually get at their own voice mail but by mistake typed the ‘##’ short cut to access the (unprotected) voice mail. Promptly he got a message (intended for the actual owner of the extension) which was by his girlfriend who steamily described various activities she would undertake with him if he could only make it home from the office on time. After that incident (of some embarrassment and plenty of laughter too admittedly) we quickly revised our companies security policy to embrace the voice mail system and the PBX.

Its also not a good idea to allow users to forward other peoples phones, make priority calls or override send-all-to-cover functions etc. The PBX offers two concepts for you to specify what phones are allowed to do.

The Class of Restriction (COR) governs the type of external calls principals are allowed to make. Some companies do not allow a secretary to make long distance calls. Directors are usually allowed to do this. This kind of setup has led to the introduction of a bridged call-appearance – the secretary has a button to get at the line of the boss in order to make the long distance call and hand the connection back to the boss.

We often found this in our PBX – a lot of extra features have crept in to augment or work around a restrictive feature instead of leaving both (the restriction and the work-arounds) out altogether. However at iXOS we are a fairly easy going place – everyone is allowed to make long distance and international calls for example so we disable most of the things the manufacturers sell you as a goodie. More restrictive companies however need these functions. In fact we also do restrict some of our phones to only make internal and emergency calls as soon as we had caught attendees of our training courses making hour long calls to the States from our training rooms.

The Class of Service (COS) governs what a phone is allowed to do. Most importantly it adjusts things like: is the phone allowed to make calls in the first place (or just receive them), if it is allowed to access the Feature access Codes and if it allowed to do special things like having console permissions. The console in this case refers to a special phone known as the attendant console – this is colloquially referred to as the switchboard. This console has special permissions defined in its COS. It can issue call redirects for all other extensions. It can override any send-all-calls

or call forwards the users may have enabled. It is allowed to re-record announcements and there is a plethora of other functions available only at the console.

Some other terminals like our sysadmin phones (AT&T CallMasters – the second to ugliest telephones in the universe after the Veeblezork Mark III found in use on Disconnectus 5) we can also have a COS giving them console permissions, so we can get through to people and act for and on behalf of the switchboard in cases of need.

Unfortunately the manufacturer of our PBX ignored some programming principles (again). As far as I am concerned I want to be able to define COS as a special class of its own and I define which functions (and function buttons) are allowed to be contained in this. Unfortunately our switch has an implicit idea of what buttons certain devices are and are not allowed to have. The console for example may not have a call-forward button. This is highly annoying because at times we want someone else to just handle incoming calls for a while.

I had to spend hours working around this problem by designating backup-consoles as a split and defining an ACD queue which only ever has one member – either the console proper or one of the backup consoles. How much easier my life would have been if I could have just defined a new console COS and designated several stations to be of that COS. In the definition of said COS I would have then allowed a call-forward button and life would have been much simpler. My workaround took ages of consulting with the manufacturer and the implementation frittered away several precious function buttons on the backup consoles.

Installation Issues

This brings us to the subject of installing the PBX – this is not merely buying the thing and plugging it in. You have to have a very good idea of the initial programming and special features you want to have implemented. You should have a good idea of the functions you want to make accessible to your users through the function keys.

In this respect we found it was a key move of utter strategic brilliance (actually it was some coincidence too) to take in the sysadmin training of the manufacturer before the installation. This way we knew about hoopy features and caveats and most importantly we spoke the language of the installing technicians. This helped a great deal and on a purely social level we got to be really friendly with them because they were happy to have a customer in the know who took the whole thing seriously.

We grossly underestimated the time (and consequently money) it took to get our phone lines sorted out by our electricians. This is required because digital lines are treated more carefully – the PBX measures

the lines' impedance and quality. If the criteria are not met, the PBX will simply disable the port. So, you must have the copper wires in shape before the PBX arrives. If the switch decides to disable 40 or 60% of your lines on the installation day then you will find out what life is like with a big problem on your hands. The monetary cost of tracing our lines, checking them out and preliminary installation (the fiber optic cables, the distribution panels and UPS wiring etc. etc.) and clean ups amounted to half of the entire hardware cost of the PBX including the phones (over \$60,000!).

Count on ordering at least 50 more phones and 20% more ports in the beginning. We thought we were at least good at mundane activities like counting things up but in the end we found we had forgotten a number of phones and also suddenly we were confronted with requests to finally put phones in storage rooms, empty offices, shacks in the swamps etc. which never had a phone before. Think hard: it might be good idea to buy that expansion sub-carrier now with your initial order when you can haggle for a massive discount.

Make a good plan of when to cut over from the old switch to the new one. Tell everyone the phones will be down – we did the cutover on a Friday afternoon when things go quiet. Think hard about which phones have to come back up quickest: the switchboard, the fax machines, the secretaries and the directors. Be sure to put the sysadmin phones online last. When you are just swapping the 134th of 260 phones you don't want to be flooded with calls of the type of "my phone is funny", "my phone doesn't work", etc. No thanks.

It took us less than ten minutes to get the switchboard back up. Fax machines and secretaries were back in business within forty-five minutes, the directors phones were on-line thirty minutes later.

Distribute the user documentation along with the new phones. Not an iota later. We also put up a web page with the most important features and the documentation of our configuration.

How are you going to get rid of the old switch and phones? We got the manufacturer of the new switch to take it all back (actually they sell the things off to Third World countries). When you have to collect more than 250 phones its a good idea to have a bunch of quarters ready and -ahem- borrow some shopping trolleys from the local supermarket. These are very handy to cart around large quantities of new and old phones.

Don't forget that an installation of this type is actually also fun! OK, it was a lot of work (eight people were busy from 3 till 11 p.m. plus all the work up front). But its only rarely that you can do something so radical and novel like giving everyone a new phone and dragging them kicking and screaming into a new digital age. We all loved and the team spirit was really good. We had a blast.

Sysadmin Support by the PBX – our Hot Line

We also use the hotline/ACD function of the PBX to help us with our sysadmin duties. We have defined a sysadmin hotline number extension (in our case 911 – and its available on a function key on every phone. (The German equivalent of 911 is a different number so there is no confusion, however 911 is a well enough known concept to be easily remembered).

One of our group is always logged into the split (usually two of us so that we can sometimes leave the desk to answer calls of nature, user requests etc.). Users call the hotline using their hot key and we take their calls. The users do not have to poll our individual extensions to get a response. Via the login process the phone system knows where to route the call to our 911 number.

So what happens if I am logged in and have to leave my seat and mere seconds after I have left the room my phone starts ringing because I have left the login button depressed? Our phone system has a feature called RONA (Ring On No Answer). If a logged in agent does not pick up the call within two rings the PBX will immediately route the call to another logged in extension at highest priority. Thus it will scan successively until it finds an agent willing to handle the call.

Note that this ringing and searching is done without the caller noticing: all they hear is some announcements and the music to fill the gaps. They hear no ringing tone and ultimately they don't know or care where their call winds up as long as we answer.

All of our Call Monsters as we call them are equipped with headsets (these became really popular with the new phone system: we now have about 30 headset users).

The sysadmins staffing the sysadmin hotline act as a dual help desk and dispatcher function. Basically any requests they can handle quickly (giving advice, kissing away tears, dishing out floppies and SCSI cables etc.) are handled immediately. Anything else is entered into the request database or escalated directly to the relevant sysadmins concerned. This setup has really reduced the interrupt rate of the other sysadmins immensely who can now get some real work done.

We change roles on demand: sometimes we have a morning and afternoon shift, sometimes one of us is dispatcher and help desk for one or two entire days, just as we deem fit. In order to keep our users informed about who is on duty (911 is of course always on duty from 8 a.m. till 6 p.m.) and may be interrupted we bought one of those "Times Square" LED displays which tells people about the admin on duty to correctly route user requests being delivered by Sneaker Net.

On the whole our sysadmin hotline has been a huge success for our group and our user community. Other departments (customer support and sales) have

now taken to wanting (and getting) their own hotlines and queues etc. so they can do similar things and achieve similar improvements.

Conclusion

When we first bought the new PBX we were wary if we needed such a complex beast. However our experiences with this installation and the ease of maintenance and the level of support we can now give is worth that initial outlay. Our users are very much happier with the phone system and the fact they see and notice that we take care of it.

Also our implementation of our sysadmin hotline shows how even small companies with even tinier sysadmin departments can make good use of advanced features of modern call centers. This is really not overkill but hopefully the shape of things to come.

Author Information

Snoopy is the local resident court jester, author and system and network admin for iXOS Software GmbH (in roughly that order). He has been with iXOS from the beginning in 1988 – before that he spent time time developing hardware and writing software for various German companies. His e-mail address is: Snoopy@munich.ixos.de.

NOTES

USENIX ASSOCIATION

USENIX is the UNIX and Advanced Computing Systems Technical and Professional Association. Since 1975 the USENIX Association has brought together the community of engineers, system administrators, scientists, and technicians working on the cutting edge of the computing world.

The USENIX Conferences have become the essential meeting grounds for the presentation and discussion of the most advanced information on new developments in all aspects of advanced computing systems.

The USENIX Association and its members are dedicated to:

- problem-solving with a practical bias,
- fostering innovation and research that works,
- rapidly communicating results of research and innovation,

- providing a neutral forum for the exercise of critical thought and the airing of technical issues.

USENIX holds an annual multi-topic technical conference, the annual Systems Administration (LISA) conference, and frequent single-topic symposia addressing topics such as UNIX security, Tcl/Tk, object-oriented technologies, networking, electronic commerce, and operating systems design – as many as ten technical meetings every year. It publishes *login.*, a bi-monthly newsletter; and proceedings for each of its conferences and symposia. It also sponsors special technical groups as well as participating in various standards efforts such as IEEE, ANSI, and ISO.

SAGE, the System Administrators Guild

SAGE, a Special Technical Group within the USENIX Association, is dedicated to the recognition and advancement of system administration as a profession. To join SAGE, you must be a member of USENIX.

SAGE activities currently include the publishing of the "Short Topics in System Administration" series, the first of which is "Job Descriptions for System Administrators", and the next two of which will be "Writing Policies for Computer Sites" and "Systems Security: A Management Perspective"; "SAGE News", a regular section in *login.*; The System Administrator Profile, an annual survey of system administrator salaries and responsibilities; co-sponsoring the LISA and SANS conferences, support of working groups; encouraging the formation of local SAGE groups; and an archive site for papers from the LISA conferences and sys admin-related documentation.

As a member of the USENIX Association/SAGE, you receive:

- Access to the papers from the USENIX Conference and Symposia proceedings, starting with 1993, via the USENIX Online Library on the World Wide Web (this includes the 1993, 1994, and 1995 LISA Conferences).
- Free subscription to *login.*, the Association's bi-monthly newsletter featuring technical articles, SAGE News, columns on tools and techniques, book and software reviews, summaries of sessions at USENIX conferences,

snitch Reports from various ANSI, IEEE, and ISO standards efforts, and much more.

- Discounts on registration for technical sessions at all USENIX conferences and symposia.
- Discounts on the purchase of proceedings from USENIX conferences and symposia.
- PGP Key signing service (available at conferences)
- Discount on the 4.4BSD Manuals plus CD-ROM published by the USENIX Association and O'Reilly & Associates, Inc.
- Discounts on the purchase of USENIX CD-ROMs.
- Special subscription rates to the periodicals UniForum Monthly, UniNews, the annual UniForum Open Systems Products Directory, and Linux Journal.
- Savings on selected titles from McGraw-Hill, The MIT Press, Prentice Hall, John Wiley & Sons, O'Reilly and Associates, and UniForum.
- Discount on all publications and software from Prime Time Freeware.
- Discount on software from BSDI, Inc.
- Right to vote on matters affecting the Association, its by-laws, election of its directors and officers.
- Right to join Special Technical Groups such as SAGE.
- SAGE members receive the following additional benefits: The most recent pamphlet in the "Short Topics in System Administration" series; the annual System Administrator Profile; the annual SAGE Calendar.

Supporting Members of the USENIX Association:

ANDATACO
Apunix Computer Services
Frame Technology Corporation
ISG Technologies, Inc.
Matsushita Electric Industrial Co., Ltd.
Motorola Research & Development

Open Market, Inc.
Shiva Corporation
Sybase, Inc.
Tandem Computers, Inc.
UUNET Technologies, Inc.

SAGE Supporting Members:

Bluestone, Inc.
Enterprise Systems Management Corp.
Great Circle Associates

Pencom Systems Inc.
Southwestern Bell
Taos Mountain

For more information about USENIX and SAGE, please contact:

The USENIX Association
2560 Ninth Street, Suite 215
Berkeley, CA 94710
WWW URL: <http://www.usenix.org>

Phone: +1 510 528-8649
Fax: +1 510 548-5738
Email: office@usenix.org

ISBN 1-880446-81-2